Practical Machine Learning - Final Project

Aleksey Kramer

February 6, 2016

Assignment

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).

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 data for this project come from this source: 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.

What you should submit The goal of your 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.

Final Project Steps

1. Load the *testing* and *training* data from online repositories supplied for the assignment
2. Using nearZeroVar function to remove sparse columns from the *testing* and *training* data sets
3. Removing the columns containing NA only from both, *training* and *testing* data set
4. Removing the last column from the *testing* data set (Result ID) and replacing it with a column called *classe* supplying the same levels of factors and populating the column with dummy data. This column would never be used, but is required for the random forest algorithm to be there.
5. Removing columns 1 through 6 from both *training* and *testing* data sets to assure that there are no discrepancies exist between *training* and *testing* data sets: factor levels are the same and classes of columns are the same.
6. Coercing classes for all fields between *training* and *testing* data frames to be the same, thus, at this stage, making *training* and *testing* data sets supplied for the assignment to be processed identically and having identical data types, factors, classes, and etc. between the data sets.
7. Sub splitting training dataset supplied for the assignment into *my_training* and *my_testing* data sets to assure that cross-validation requirement for the assignment is addressed.

8. Train the treeFit model using *my_training* data set (classification tree) and run predictions using *my_testing* data set. The accuracy of the model is in low seventy percent.
9. Train the rfFit model using *my_training* data set (random forest algorythm) and run predictions using *my_testing* data set. The accuracy of the mode is better than 99%. Thus, rfFit should be used for prediction on the *testing* data set supplied for the assignment.
10. Calculate prediction using rfFit model build using *my_training* data set and *testing* data set supplied for the assignment. Print out results. Use the results to populate the quiz.
11. Out of sample erro was calculated using *my_testing* data set instead of *testing* data set. The problem was that *classe* column was not provided as part of the data set, but I used *classe* variable. The general principle of calculating out of sample error would be exaclty the same but using *testing* data set provided for the assignment.

Code and explanations Load librarires and set working directory

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.2.3
```

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

```
## Warning: package 'ggplot2' was built under R version 3.2.2
```

```
library(rpart)
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.2.3
```

```
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
```

```
set.seed(212121)
```

Download data files if not already downloaded

```
# Set appropriate working directory
# setwd('C:\\Users\\db345c\\Desktop\\Practical Machine Learning\\Final_Project')
setwd('C:\\Users\\Aleksey\\Documents\\School\\coursera\\Practical_Machine_Learning\\Final_Project')

# Check if the files exist locally, if not - download
training.url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testing.url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

if(!file.exists("pml-training.csv")) {
    download.file(training.url, "pml-training.csv")
}

if(!file.exists("pml-testing.csv")){
    download.file(testing.url, "pml-testing.csv")
}
```

Loading data files into memory

```
# Load training and testing data sets
training <- read.csv("pml-training.csv", na.strings = c("NA","", "#DIV/0!"))
testing <- read.csv("pml-testing.csv", na.strings = c("NA","", "#DIV/0!"))

# cleanup unused variables
rm(training.url, testing.url)
```

Prepare the data to be consumed by the prediction algorythms

```
# identifying sparsely populated columns using nearZeroVar function
# from testing and training data sets
# Identical changes need to be made to testing and training data sets
to_skip <- nearZeroVar(training)
training <- training[, -to_skip]
testing <- testing[, -to_skip]

# cleanup unused variables
rm(to_skip)

# removing columns containing NA's only
# Identical changes need to be made to testing and training data sets
col_lst <- colSums(is.na(training)) == 0
training <- training[, col_lst]
testing <- testing[, col_lst]

# adjusting data - remove column 59 from the testing data (not needed)
testing <- testing[, -c(59)]

# creating simple dummy data with 5 levels identical to those in training data set
# iterate throug the data frame using sample function to select a sample of size 1 from
# vector called levels. The classe variable was not provided as part of the testing
# data set, this is why I have to add a dummy classe variable.
train_levels <- levels(training$classe)
v <- vector(mode="character", length=0)
for (i in testing[,1]) {
  x <- sample(train_levels, size=1, replace = TRUE)
  v <- c(v, x)
}

# apppending dummy data (as factor) with the same factors and number of factors as in the training data
# as well as with the same name (simply for convenience)
testing$classe <- as.factor(v)

# Remove remove index, factor, and several date columns from training and testing data set
testing <- testing[, -c(1:6)]
training <- training[, -c(1:6)]

# cleanup unused variables
rm(col_lst, v, train_levels, x)
```

Explore data frames to identify discrepancies in field's class types

```r
# Explore training and testing data to assure that all fields in both training and testing
# data sets have identical classes for each of the fields.
str(training)
```

```
## 'data.frame':    19622 obs. of  53 variables:
##  $ 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 ...
##  $ 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 ...
##  $ 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 ...
##  $ 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_x    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ gyros_dumbbell_y    : num  -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 ...
##  $ gyros_dumbbell_z    : num  0 0 0 -0.02 0 0 0 0 0 0 ...
##  $ accel_dumbbell_x    : int  -234 -233 -232 -232 -233 -234 -232 -234 -232 -235 ...
##  $ accel_dumbbell_y    : int  47 47 46 48 48 48 47 46 47 48 ...
##  $ accel_dumbbell_z    : int  -271 -269 -270 -269 -270 -269 -270 -272 -269 -270 ...
##  $ magnet_dumbbell_x   : int  -559 -555 -561 -552 -554 -558 -551 -555 -549 -558 ...
##  $ magnet_dumbbell_y   : int  293 296 298 303 292 294 295 300 292 291 ...
##  $ magnet_dumbbell_z   : num  -65 -64 -63 -60 -68 -66 -70 -74 -65 -69 ...
##  $ 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_y     : num  0 0 -0.02 -0.02 0 -0.02 0 -0.02 0 0 ...
##  $ 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_y     : int  203 203 204 206 206 203 205 205 204 205 ...
```

```
##  $ 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 ...
```

```
str(testing)
```

```
## 'data.frame':    20 obs. of  53 variables:
##  $ roll_belt           : num  123 1.02 0.87 125 1.35 -5.92 1.2 0.43 0.93 114 ...
##  $ pitch_belt          : num  27 4.87 1.82 -41.6 3.33 1.59 4.44 4.15 6.72 22.4 ...
##  $ yaw_belt            : num  -4.75 -88.9 -88.5 162 -88.6 -87.7 -87.3 -88.5 -93.7 -13.1 ...
##  $ total_accel_belt    : int  20 4 5 17 3 4 4 4 4 18 ...
##  $ gyros_belt_x        : num  -0.5 -0.06 0.05 0.11 0.03 0.1 -0.06 -0.18 0.1 0.14 ...
##  $ gyros_belt_y        : num  -0.02 -0.02 0.02 0.11 0.02 0.05 0 -0.02 0 0.11 ...
##  $ gyros_belt_z        : num  -0.46 -0.07 0.03 -0.16 0 -0.13 0 -0.03 -0.02 -0.16 ...
##  $ accel_belt_x        : int  -38 -13 1 46 -8 -11 -14 -10 -15 -25 ...
##  $ accel_belt_y        : int  69 11 -1 45 4 -16 2 -2 1 63 ...
##  $ accel_belt_z        : int  -179 39 49 -156 27 38 35 42 32 -158 ...
##  $ magnet_belt_x       : int  -13 43 29 169 33 31 50 39 -6 10 ...
##  $ magnet_belt_y       : int  581 636 631 608 566 638 622 635 600 601 ...
##  $ magnet_belt_z       : int  -382 -309 -312 -304 -418 -291 -315 -305 -302 -330 ...
##  $ roll_arm            : num  40.7 0 0 -109 76.1 0 0 0 -137 -82.4 ...
##  $ pitch_arm           : num  -27.8 0 0 55 2.76 0 0 0 11.2 -63.8 ...
##  $ yaw_arm             : num  178 0 0 -142 102 0 0 0 -167 -75.3 ...
##  $ total_accel_arm     : int  10 38 44 25 29 14 15 22 34 32 ...
##  $ gyros_arm_x         : num  -1.65 -1.17 2.1 0.22 -1.96 0.02 2.36 -3.71 0.03 0.26 ...
##  $ gyros_arm_y         : num  0.48 0.85 -1.36 -0.51 0.79 0.05 -1.01 1.85 -0.02 -0.5 ...
##  $ gyros_arm_z         : num  -0.18 -0.43 1.13 0.92 -0.54 -0.07 0.89 -0.69 -0.02 0.79 ...
##  $ accel_arm_x         : int  16 -290 -341 -238 -197 -26 99 -98 -287 -301 ...
##  $ accel_arm_y         : int  38 215 245 -57 200 130 79 175 111 -42 ...
##  $ accel_arm_z         : int  93 -90 -87 6 -30 -19 -67 -78 -122 -80 ...
##  $ magnet_arm_x        : int  -326 -325 -264 -173 -170 396 702 535 -367 -420 ...
##  $ magnet_arm_y        : int  385 447 474 257 275 176 15 215 335 294 ...
##  $ magnet_arm_z        : int  481 434 413 633 617 516 217 385 520 493 ...
##  $ roll_dumbbell       : num  -17.7 54.5 57.1 43.1 -101.4 ...
##  $ pitch_dumbbell      : num  25 -53.7 -51.4 -30 -53.4 ...
##  $ yaw_dumbbell        : num  126.2 -75.5 -75.2 -103.3 -14.2 ...
##  $ total_accel_dumbbell: int  9 31 29 18 4 29 29 29 3 2 ...
##  $ gyros_dumbbell_x    : num  0.64 0.34 0.39 0.1 0.29 -0.59 0.34 0.37 0.03 0.42 ...
##  $ gyros_dumbbell_y    : num  0.06 0.05 0.14 -0.02 -0.47 0.8 0.16 0.14 -0.21 0.51 ...
##  $ gyros_dumbbell_z    : num  -0.61 -0.71 -0.34 0.05 -0.46 1.1 -0.23 -0.39 -0.21 -0.03 ...
##  $ accel_dumbbell_x    : int  21 -153 -141 -51 -18 -138 -145 -140 0 -7 ...
##  $ accel_dumbbell_y    : int  -15 155 155 72 -30 166 150 159 25 -20 ...
##  $ accel_dumbbell_z    : int  81 -205 -196 -148 -5 -186 -190 -191 9 7 ...
##  $ magnet_dumbbell_x   : int  523 -502 -506 -576 -424 -543 -484 -515 -519 -531 ...
##  $ magnet_dumbbell_y   : int  -528 388 349 238 252 262 354 350 348 321 ...
##  $ magnet_dumbbell_z   : int  -56 -36 41 53 312 96 97 53 -32 -164 ...
##  $ roll_forearm        : num  141 109 131 0 -176 150 155 -161 15.5 13.2 ...
##  $ pitch_forearm       : num  49.3 -17.6 -32.6 0 -2.16 1.46 34.5 43.6 -63.5 19.4 ...
##  $ yaw_forearm         : num  156 106 93 0 -47.9 89.7 152 -89.5 -139 -105 ...
##  $ total_accel_forearm : int  33 39 34 43 24 43 32 47 36 24 ...
##  $ gyros_forearm_x     : num  0.74 1.12 0.18 1.38 -0.75 -0.88 -0.53 0.63 0.03 0.02 ...
##  $ gyros_forearm_y     : num  -3.34 -2.78 -0.79 0.69 3.1 4.26 1.8 -0.74 0.02 0.13 ...
```

```
##  $ gyros_forearm_z     : num  -0.59 -0.18 0.28 1.8 0.8 1.35 0.75 0.49 -0.02 -0.07 ...
##  $ accel_forearm_x     : int  -110 212 154 -92 131 230 -192 -151 195 -212 ...
##  $ accel_forearm_y     : int  267 297 271 406 -93 322 170 -331 204 98 ...
##  $ accel_forearm_z     : int  -149 -118 -129 -39 172 -144 -175 -282 -217 -7 ...
##  $ magnet_forearm_x    : int  -714 -237 -51 -233 375 -300 -678 -109 0 -403 ...
##  $ magnet_forearm_y    : int  419 791 698 783 -787 800 284 -619 652 723 ...
##  $ magnet_forearm_z    : int  617 873 783 521 91 884 585 -32 469 512 ...
##  $ classe              : Factor w/ 5 levels "A","B","C","D",..: 5 1 2 3 3 1 3 5 4 1 ...
```

Coherce field classes to be the same between two data frames: *testing* and *training*

```
# coherce training and testing data to the same data types and fix factors in testing
for (i in 1:length(testing)) {
  for(j in 1:length(training)) {
    if( length(grep(names(training[i]), names(testing)[j])) == 1)  {
      class(testing[j]) <- class(training[i])
    }
  }
}

# fix coersion that failed in previous step (this is required!!!)
for (i in 1:length(testing)) {
  if (class(training[, i]) != class(testing[, i])) {
    ### try fixing coersion, so testing and training data frames match
    class(training[, i]) <- class(testing[, i])
  }
}
```

Split *training* data set into *my_training* and *my_testing* data sets.

```
# Partition data into two two sets - training and testing
# Identical changes need to be made to testing and training data sets
inTrain <- createDataPartition(y=training$classe, p=0.6, list=FALSE)
my_training <- training[inTrain,]
my_testing <- training[-inTrain,]

# cleanup unused variables
rm(inTrain, i, j)
```

Generate tree and run prediction. Accuracy is quite low.

```
# Build decision tree model and run predictions
set.seed(212121)
treeFit <- rpart(classe ~., data=my_training, method="class")
treePredicted <- predict(treeFit, my_testing, type="class")
# print confusion matrix for treePredicted vs. my_testing$classe
confusionMatrix(treePredicted, my_testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1973  215   27   57   29
```

```
##          B   75  836  116  109  106
##          C   66  206 1110  206  182
##          D   81  120   80  827   87
##          E   37  141   35   87 1038
##
## Overall Statistics
##
##                Accuracy : 0.7372
##                  95% CI : (0.7273, 0.7469)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6675
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8840   0.5507   0.8114   0.6431   0.7198
## Specificity            0.9416   0.9358   0.8981   0.9439   0.9532
## Pos Pred Value         0.8575   0.6731   0.6271   0.6921   0.7758
## Neg Pred Value         0.9533   0.8967   0.9575   0.9310   0.9379
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2515   0.1066   0.1415   0.1054   0.1323
## Detection Prevalence   0.2933   0.1583   0.2256   0.1523   0.1705
## Balanced Accuracy      0.9128   0.7433   0.8548   0.7935   0.8365
```

Generate Random Forest and run prediction. Accuracy is more than 99%

```
rfFit <- randomForest(classe ~., data=my_training)
rfPredicted <- predict(rfFit, my_testing, type="class")
confusionMatrix(rfPredicted, my_testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232   13    0    0    0
##          B    0 1502    9    0    0
##          C    0    3 1359   12    0
##          D    0    0    0 1273    3
##          E    0    0    0    1 1439
##
## Overall Statistics
##
##                Accuracy : 0.9948
##                  95% CI : (0.9929, 0.9962)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9934
##  Mcnemar's Test P-Value : NA
##
```

```
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9895   0.9934   0.9899   0.9979
## Specificity            0.9977   0.9986   0.9977   0.9995   0.9998
## Pos Pred Value         0.9942   0.9940   0.9891   0.9976   0.9993
## Neg Pred Value         1.0000   0.9975   0.9986   0.9980   0.9995
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1914   0.1732   0.1622   0.1834
## Detection Prevalence   0.2861   0.1926   0.1751   0.1626   0.1835
## Balanced Accuracy      0.9988   0.9940   0.9956   0.9947   0.9989
```

Printing predictions for the test data set supplied for the final project

```
# printing the answer for the quiz
finalAnswer <- predict(rfFit, testing, type="class")
print(finalAnswer)
```

```
##  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
```

Caclulating out of sample eror

```
# The classe variable was not provided as a part of the testing data set supplied
# for this assignment, but it was recommended to use classe variable to predict on.
# The principle of calculating out of sample error would be the same, but with
# testing data set (if classe variable was provied). I used my_training data set.
# I expect out of sample error to be slightly larger by a percent or two.
error_rate = sum((my_testing$classe != rfPredicted) / length(rfPredicted))
print(paste0("The out of sample error rate is: ", error_rate))
```

```
## [1] "The out of sample error rate is: 0.00522559265867958"
```