Practical Machine Learning Project

Dhrumil Dalal

December 27, 2015

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.

Goal

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.

The information is available from the website here: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)

Step 0 - Define Problem statement

As indicated in the intorduction, the goal of the project is to determine the quality of the exercise. The quality of the exercise is determined by the variable “classe”.

Step 1 - Obtain Data

The training and test data for this project are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv)

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)

The original source of the data is: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har).

```
################################################################################
########################
## This function is responsible for
## 1. Downloading the file from the source URL
## 2. If the data folder does not exist then create it
## 3. Copy the files to the this folder
getDataFiles <- function(filesDirectory)
{
  setwd(".")
  if (!file.exists(filesDirectory))
  {
    dir.create(filesDirectory)

    ModelTestDataUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml
-ModelTestDataing.csv"
    ModelDevDataUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-
ModelDevDataing.csv"
    ModelDevDataFile <- "ModelDevData.csv"
    ModelTestDataFile <- "ModelTestData.csv"
    ModelDevDataFilePath <- paste(filesDirectory, ModelDevDataFile, sep = "/")
    ModelTestDataFilePath <- paste(filesDirectory, ModelTestDataFile, sep =
"/")
    download.file(ModelDevDataUrl, destfile = ModelDevDataFilePath)
    download.file(ModelTestDataUrl, destfile = ModelTestDataFilePath)
    ModelDevDataing <- read.csv(ModelDevDataFilePath, na.strings=c("NA","#DIV/
0!",""))
    ModelTestDataing <- read.csv(ModelTestDataFilePath, na.strings=c("NA","#DI
V/0!",""))
  }
}
################################################################################
########################
```

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

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

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

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

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

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

Step 2 - Basic EDA

Once the problem is defined and we have the dataset with us, the next step is to explore the data. In a real world scenario comprehensive and extensive EDA needs to be done, for the purpose of this project we will be undertaking only the baisc EDA. We examine various columns using the summary() function.

```
getDataFiles("./data")

training <- read.csv("./data/train.csv")
testing <- read.csv("./data/test.csv")
summary(training)
head(training)
```

```
getDataFiles("./data")

training <- read.csv("./data/train.csv")
testing <- read.csv("./data/test.csv")
```

We observe following

1. There are total of 155 columns in the dataset.
2. First 5 columns in the dataset downloaded are not useful for analysis purpose as they contain user information and audit columns.
3. There are columns in the dataset where more than 50% rows have the values of NULL or NA.

Step 3 - Training Data Preparation

We remove these columns from the training dataset. We also remove the columns from the testing dataset.

```
## Remove first 5 colums from the training set
dimTr <- dim(training);
training <- training[,6:dimTr[2]]
dimTr <- dim(training);

## Remove first 5 colums from the testing set
dimTest <- dim(testing);
testing <- testing[,6:dimTest[2]]
dimTest <- dim(testing);
```

Step 4 - Feature Selection

Next we determine the that columns have impact on the output columns. We do this by using NearZeroValue function. Then we perform similar operation on the testing set.

```
uselessCol <- nearZeroVar(training, saveMetrics=TRUE)$nzv
for ( i in 1:dimTr[2]  )
{
  if (sum(is.na(training[,i]))/dimTr[1] > 0.8)
  {
    uselessCol[i] <- TRUE

  }

}
training <- training[, uselessCol==FALSE]
testing <- testing [, uselessCol==FALSE]
```

At this poin we are left with the columns that have significant impact on the output variable. The list of the input variables can be seen using the head() function

```
head(training)
```

```
##   num_window roll_belt pitch_belt yaw_belt total_accel_belt gyros_belt_x
## 1         11      1.41       8.07    -94.4                3         0.00
## 2         11      1.41       8.07    -94.4                3         0.02
## 3         11      1.42       8.07    -94.4                3         0.00
## 4         12      1.48       8.05    -94.4                3         0.02
## 5         12      1.48       8.07    -94.4                3         0.02
## 6         12      1.45       8.06    -94.4                3         0.02
##   gyros_belt_y gyros_belt_z accel_belt_x accel_belt_y accel_belt_z
## 1         0.00        -0.02          -21            4           22
## 2         0.00        -0.02          -22            4           22
## 3         0.00        -0.02          -20            5           23
## 4         0.00        -0.03          -22            3           21
## 5         0.02        -0.02          -21            2           24
## 6         0.00        -0.02          -21            4           21
##   magnet_belt_x magnet_belt_y magnet_belt_z roll_arm pitch_arm yaw_arm
## 1            -3           599          -313     -128      22.5    -161
## 2            -7           608          -311     -128      22.5    -161
## 3            -2           600          -305     -128      22.5    -161
## 4            -6           604          -310     -128      22.1    -161
## 5            -6           600          -302     -128      22.1    -161
## 6             0           603          -312     -128      22.0    -161
##   total_accel_arm gyros_arm_x gyros_arm_y gyros_arm_z accel_arm_x
## 1              34        0.00        0.00       -0.02        -288
## 2              34        0.02       -0.02       -0.02        -290
## 3              34        0.02       -0.02       -0.02        -289
## 4              34        0.02       -0.03        0.02        -289
## 5              34        0.00       -0.03        0.00        -289
## 6              34        0.02       -0.03        0.00        -289
##   accel_arm_y accel_arm_z magnet_arm_x magnet_arm_y magnet_arm_z
## 1         109        -123         -368          337          516
## 2         110        -125         -369          337          513
## 3         110        -126         -368          344          513
## 4         111        -123         -372          344          512
## 5         111        -123         -374          337          506
## 6         111        -122         -369          342          513
##   roll_dumbbell pitch_dumbbell yaw_dumbbell total_accel_dumbbell
## 1      13.05217      -70.49400    -84.87394                   37
## 2      13.13074      -70.63751    -84.71065                   37
## 3      12.85075      -70.27812    -85.14078                   37
## 4      13.43120      -70.39379    -84.87363                   37
## 5      13.37872      -70.42856    -84.85306                   37
## 6      13.38246      -70.81759    -84.46500                   37
##   gyros_dumbbell_x gyros_dumbbell_y gyros_dumbbell_z accel_dumbbell_x
## 1                0            -0.02             0.00             -234
## 2                0            -0.02             0.00             -233
## 3                0            -0.02             0.00             -232
## 4                0            -0.02            -0.02             -232
## 5                0            -0.02             0.00             -233
```

```
## 6                 0             -0.02             0.00              -234
##   accel_dumbbell_y accel_dumbbell_z magnet_dumbbell_x magnet_dumbbell_y
## 1               47             -271              -559               293
## 2               47             -269              -555               296
## 3               46             -270              -561               298
## 4               48             -269              -552               303
## 5               48             -270              -554               292
## 6               48             -269              -558               294
##   magnet_dumbbell_z roll_forearm pitch_forearm yaw_forearm
## 1               -65         28.4         -63.9        -153
## 2               -64         28.3         -63.9        -153
## 3               -63         28.3         -63.9        -152
## 4               -60         28.1         -63.9        -152
## 5               -68         28.0         -63.9        -152
## 6               -66         27.9         -63.9        -152
##   total_accel_forearm gyros_forearm_x gyros_forearm_y gyros_forearm_z
## 1                  36            0.03            0.00           -0.02
## 2                  36            0.02            0.00           -0.02
## 3                  36            0.03           -0.02            0.00
## 4                  36            0.02           -0.02            0.00
## 5                  36            0.02            0.00           -0.02
## 6                  36            0.02           -0.02           -0.03
##   accel_forearm_x accel_forearm_y accel_forearm_z magnet_forearm_x
## 1             192             203            -215              -17
## 2             192             203            -216              -18
## 3             196             204            -213              -18
## 4             189             206            -214              -16
## 5             189             206            -214              -17
## 6             193             203            -215               -9
##   magnet_forearm_y magnet_forearm_z classe
## 1              654              476      A
## 2              661              473      A
## 3              658              469      A
## 4              658              469      A
## 5              655              473      A
## 6              660              478      A
```

Now we prepare the final set that will be used for training the model and perform in-DataSet testing of the model. The training set is split in 2 parts; dataset used to train the model and dataset to perform in-DataSet testing. We use

60% of the data for training the model and

40% of data for in-DataSet testing

In order to make the results reproducible we set the seed to 786

```
set.seed(786)
inTrain <- createDataPartition(y=training$classe, p=0.6, list=FALSE)
myTraining <- training[inTrain, ]
myTesting <- training[-inTrain, ]
```

Step 5 - Model Selection

Model selection consists of several steps. In this process we need to try out various algorithms. As we select an algorithm we also ensure there is no over-fitting in the model.

Step 5a - Avoid Overfitting

We prepare a control-set, where we specify the method to be used as 10 fold cross validation. We also specify that we want to use PCA (Principal Component Analysis) as pre-processing option.

```
tc <- trainControl(method = "cv", number = 10, verboseIter=FALSE , preProcOptio
ns="pca", allowParallel=TRUE)
```

Step 5b - Model Training

Random Forest and Recursive Partitioning algorithms are used to train model. We observe that

1. For Random Forest algorithm even with 60% data used for training and 10 folds for Cross validation; the accuracy is fairly consistent acorss all 10 folds.
2. For Recursive Partitioning the accuracy is failry consistent acorss all 10 folds.
3. The accuracy of Random forest is very high 99.68%
4. The accuracy of Recursive Partitioning is very poor 53.17%

```
rf <- train(classe ~ ., data = myTraining, method = "rf", trControl= tc)
rf$resample
rf$results


rpart <- train(classe ~ ., data = myTraining, method = "rpart", trControl= tc)
rpart$resample
rpart$results
```

```
## [1] "The cross validation result using Random Forest is "
```

```
##     Accuracy     Kappa Resample
## 1  0.9932088 0.9914068   Fold02
## 2  0.9966073 0.9957090   Fold01
## 3  0.9966044 0.9957059   Fold04
## 4  0.9957519 0.9946288   Fold03
## 5  0.9983022 0.9978523   Fold06
## 6  0.9983022 0.9978523   Fold05
## 7  0.9974490 0.9967737   Fold08
## 8  0.9974511 0.9967756   Fold07
## 9  0.9983022 0.9978525   Fold10
## 10 0.9966015 0.9957001   Fold09
```

```
##   mtry  Accuracy     Kappa  AccuracySD     KappaSD
## 1    2 0.9937169 0.9920513 0.002806413 0.003551559
## 2   27 0.9968581 0.9960257 0.001552457 0.001964298
## 3   53 0.9947354 0.9933407 0.001687614 0.002134328
```

```
## [1] "The cross validation result using Recursive Partitioning is "
```

```
##     Accuracy     Kappa Resample
## 1  0.5382653 0.4083510   Fold02
## 2  0.5493197 0.4304455   Fold01
## 3  0.5385920 0.3993099   Fold03
## 4  0.5360475 0.4052204   Fold06
## 5  0.5420561 0.4203443   Fold05
## 6  0.4753820 0.3145856   Fold04
## 7  0.4859813 0.3272545   Fold07
## 8  0.5611205 0.4449402   Fold10
## 9  0.5463042 0.4260602   Fold09
## 10 0.5445293 0.4165529   Fold08
```

```
##           cp  Accuracy      Kappa AccuracySD    KappaSD
## 1 0.03844328 0.5317598 0.39930645 0.02795339 0.04345338
## 2 0.06059168 0.4042199 0.18939918 0.05716678 0.09686968
## 3 0.11734694 0.3152208 0.04743643 0.03992173 0.06127635
```

Step 5c - Model Validation (In- DataSet Testing)

Next we perform in-Data set testing. Again we observe that

1. For Random Forest the accuracy for In-DataSet testing (99.58%) is fairly consistent with each folds in the cross validation (99.68%).
2. For Recurssive Partitioning there is slight drop in the accuracy of In-DataSet testing (49.24%) from average accuracy in cross validation (53.17%)

```
confusionMatrix(predict(rf,myTesting),myTesting$classe)

confusionMatrix(predict(rpart,myTesting),myTesting$classe)
```

```
## [1] "The Accuracy for In-DataSet testing using Random Forset is "
```

```
## Accuracy
## 0.995794
```

```
## [1] "The Accuracy for In-DataSet testing using Reccursive Partition is "
```

```
##  Accuracy
## 0.4923528
```

```
## [1] "The confusion matrix for In-DataSet testing using Random Forest is "
```

```
## $table
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    7    0    0    0
##          B    0 1508    5    0    0
##          C    0    3 1359    8    0
##          D    0    0    4 1277    5
##          E    0    0    0    1 1437
```

```
## [1] "The confusion matrix for In-DataSet testing using Reccursive Partition
is "
```

```
## $table
##           Reference
## Prediction    A    B    C    D    E
##          A 2006  640  653  563  201
##          B   39  515   42  223  205
##          C  154  363  673  500  367
##          D    0    0    0    0    0
##          E   33    0    0    0  669
```

The error in the predicting function is 1 - Accuracy.
That will be
1 - 0.995 = 0.005 (Random Forest)
1 - 0.492 = 0.508 (Recursive Trees)

Step 6 - Model Validation (Out-DataSet Testing)

Next we perform out-DataSet testing.

```
RFPrediction <- predict(rf,testing)
print("Prediction with Random Forest")
RFPrediction


RPartPrediction <- predict(rpart,testing)
print("Prediction with Recurcive Partitioning")
RPartPrediction
```

```
## [1] "Prediction with Random Forest"
```

```
##  [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
```

```
## [1] "Prediction with Recurcive Partitioning"
```

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

Step 6 - Conclusion For the given data set Random Forest performs better than Recursive Partitioning. We will be using the RFPrediction model for next part of the submission