Practical Machine Language

Carlos Martinez

March 2, 2016

Practical Machine Learning

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.

Data:

```
Training data:
https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv
Test data:
https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv
```

Prediction:

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.

1. Load library's and read in train and test Data

    ```
    library(caret)
    pml.train <- read.csv("pml_training.csv")
    pml.test <- read.csv("pml_testing.csv")
    ```

2. We next examine the test data : `pml.test`

```
a. We search for data without NA's
b. Those values without NA's become possible predictor's
c. Search for belt , arm and dumbbell keywords.
```

```
tmpMissing <- sapply(pml.test, function (x) any(is.na(x) | x == ""))
tmpPredictor <- !tmpMissing & grepl("belt|[^(fore)]arm|dumbbell|forearm", names(tmpMissing))
Predictors <- names(tmpMissing)[tmpPredictor]
Predictors
```

```
##  [1] "roll_belt"            "pitch_belt"           "yaw_belt"
##  [4] "total_accel_belt"     "gyros_belt_x"         "gyros_belt_y"
##  [7] "gyros_belt_z"         "accel_belt_x"         "accel_belt_y"
## [10] "accel_belt_z"         "magnet_belt_x"        "magnet_belt_y"
## [13] "magnet_belt_z"        "roll_arm"             "pitch_arm"
## [16] "yaw_arm"              "total_accel_arm"      "gyros_arm_x"
## [19] "gyros_arm_y"          "gyros_arm_z"          "accel_arm_x"
## [22] "accel_arm_y"          "accel_arm_z"          "magnet_arm_x"
## [25] "magnet_arm_y"         "magnet_arm_z"         "roll_dumbbell"
## [28] "pitch_dumbbell"       "yaw_dumbbell"         "total_accel_dumbbell"
## [31] "gyros_dumbbell_x"     "gyros_dumbbell_y"     "gyros_dumbbell_z"
## [34] "accel_dumbbell_x"     "accel_dumbbell_y"     "accel_dumbbell_z"
## [37] "magnet_dumbbell_x"    "magnet_dumbbell_y"    "magnet_dumbbell_z"
## [40] "roll_forearm"         "pitch_forearm"        "yaw_forearm"
## [43] "total_accel_forearm"  "gyros_forearm_x"      "gyros_forearm_y"
## [46] "gyros_forearm_z"      "accel_forearm_x"      "accel_forearm_y"
## [49] "accel_forearm_z"      "magnet_forearm_x"     "magnet_forearm_y"
## [52] "magnet_forearm_z"
```

3. Next extract the `Predictors` and `classe` variables from pml.train

```
pml.train <- pml.train[, c("classe", Predictors)]
dim(pml.train)
```

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

4.We are now ready to work with the pml.train dataset. We split the dataset into proportions of 60/40 : training/testing

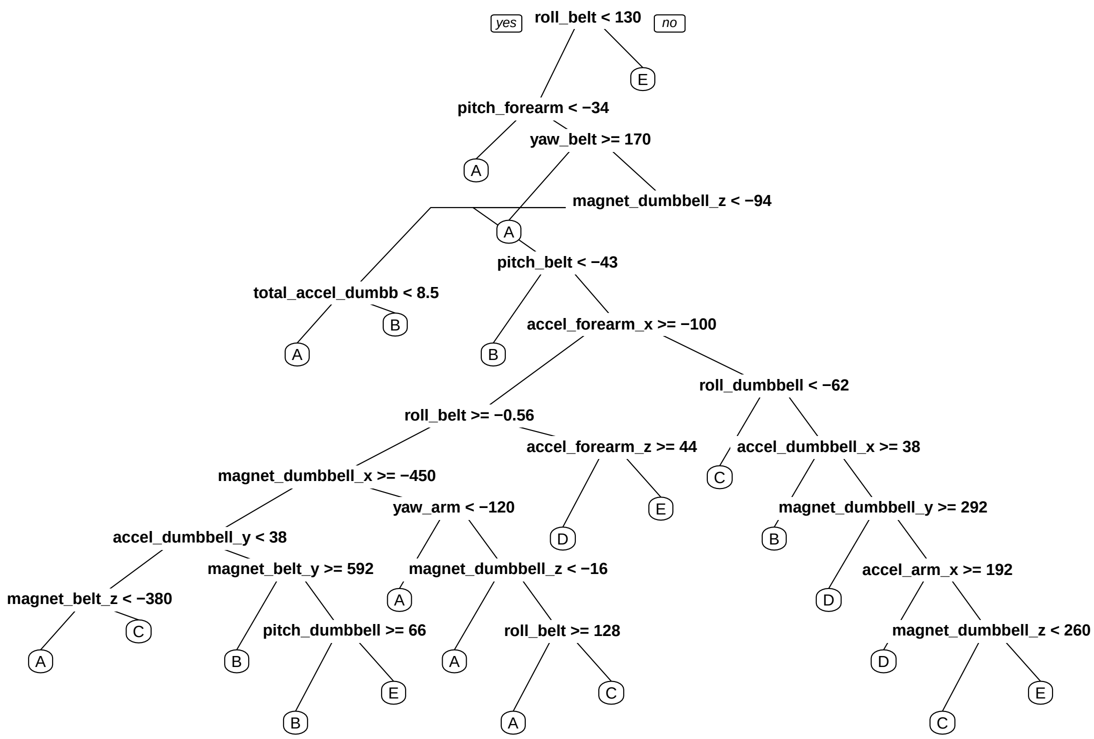
```
set.seed(12345)
trainIndex <- createDataPartition(y=pml.train$classe, p=0.6, list=FALSE)
training <- pml.train[trainIndex, ]
testing <- pml.train[-trainIndex, ]
dim(training); dim(testing)
```

```
## [1] 11776    53
```

```
## [1] 7846   53
```

Classification Tree diagram with prediction

```r
library(rpart)
rpFit <- rpart(classe ~ ., data=training, method = "class")
library(rpart.plot)
prp(rpFit)
```




```r
predictz <- predict(rpFit, testing, type = "class")
```

Now for classification tree Confusion Matrix

```r
confusionMatrix(predictz, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1879  260   30   69   66
##          B   56  759   88   34   54
##          C  105  340 1226  354  234
##          D  155  132   23  807   57
##          E   37   27    1   22 1031
##
## Overall Statistics
##
##                Accuracy : 0.7267
##                  95% CI : (0.7167, 0.7366)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6546
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8418  0.50000   0.8962   0.6275   0.7150
## Specificity            0.9243  0.96334   0.8405   0.9441   0.9864
## Pos Pred Value         0.8155  0.76589   0.5427   0.6874   0.9222
## Neg Pred Value         0.9363  0.88928   0.9746   0.9282   0.9389
## Prevalence             0.2845  0.19347   0.1744   0.1639   0.1838
## Detection Rate         0.2395  0.09674   0.1563   0.1029   0.1314
## Detection Prevalence   0.2937  0.12631   0.2879   0.1496   0.1425
## Balanced Accuracy      0.8831  0.73167   0.8684   0.7858   0.8507
```

Random Forest with prediction in-sample error

```
library(randomForest)
rfFit <- randomForest(classe ~. , data=training)
predictx <- predict(rfFit, testing, type = "class")
```

Now for random forest Confusion Matrix

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2229    7    0    0    0
##          B    3 1505    5    0    0
##          C    0    6 1363   16    2
##          D    0    0    0 1268    4
##          E    0    0    0    2 1436
##
## Overall Statistics
##
##                Accuracy : 0.9943
##                  95% CI : (0.9923, 0.9958)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9927
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9987   0.9914   0.9963   0.9860   0.9958
## Specificity            0.9988   0.9987   0.9963   0.9994   0.9997
## Pos Pred Value         0.9969   0.9947   0.9827   0.9969   0.9986
## Neg Pred Value         0.9995   0.9979   0.9992   0.9973   0.9991
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2841   0.1918   0.1737   0.1616   0.1830
## Detection Prevalence   0.2850   0.1928   0.1768   0.1621   0.1833
## Balanced Accuracy      0.9987   0.9951   0.9963   0.9927   0.9978
```

Comparing both , Random Forest shows the best promise.

We will use Random Forest for predictions