Machine Learning

Peter Skelton

24 September 2015

Synopsis

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. 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, the goal will be to use data from accelerators on the belt, forearm, arm, and dumbbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. This paper will outline building a machine learning algorithm to predict in which of the 5 different ways the activity performed.

The final model had an out of sample error of 0.7% on the testing data split. It correctly predicted all 20 cases of the test set of data.

More information on the data set is available from the website here: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har) (see the section on the Weight Lifting Exercise Dataset)

Data Processing

First we will need to download and then read in the data set.

```
traindataurl <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.
csv"
testdataurl <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.cs
v"

traindatapath <- "data\\pml-training.csv"
testdatapath <- "data\\pml-testing.csv"

if ( !file.exists( traindatapath ) ){
    download.file( traindataurl, destfile = traindatapath )
    download.file( testdataurl, destfile = testdatapath )
}

training <- read.csv( traindatapath )
testing <- read.csv( testdatapath )
```

Take a look at the data to see what we are dealing with.

```
dim( training )
```

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

```
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 1
323084232 1323084232 1323084232 1323084232 1323084232 1323084232 ...
##  $ raw_timestamp_part_2     : int  788290 808298 820366 120339 196328 304277 368
296 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       : Factor w/ 397 levels "","-0.016850",..: 1 1 1 1 1
1 1 1 1 1 ...
##  $ kurtosis_picth_belt      : Factor w/ 317 levels "","-0.021887",..: 1 1 1 1 1
1 1 1 1 1 ...
##  $ kurtosis_yaw_belt        : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1
 1 ...
##  $ skewness_roll_belt       : Factor w/ 395 levels "","-0.003095",..: 1 1 1 1 1
1 1 1 1 1 ...
##  $ skewness_roll_belt.1     : Factor w/ 338 levels "","-0.005928",..: 1 1 1 1 1
1 1 1 1 1 ...
##  $ skewness_yaw_belt        : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1
 1 ...
##  $ 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             : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1
 1 1 1 1 1 ...
##  $ 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             : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1
 1 1 1 1 1 ...
##  $ 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       : Factor w/ 4 levels "","#DIV/0!","0.00",..: 1 1 1 1
 1 1 1 1 1 1 ...
##  $ 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     : Factor w/ 330 levels "","-0.02438",..: 1 1 1 1 1 1
 1 1 1 1 ...
##  $ kurtosis_picth_arm    : Factor w/ 328 levels "","-0.00484",..: 1 1 1 1 1 1
 1 1 1 1 ...
##  $ kurtosis_yaw_arm      : Factor w/ 395 levels "","-0.01548",..: 1 1 1 1 1 1
 1 1 1 1 ...
##  $ skewness_roll_arm     : Factor w/ 331 levels "","-0.00051",..: 1 1 1 1 1 1
 1 1 1 1 ...
##  $ skewness_pitch_arm    : Factor w/ 328 levels "","-0.00184",..: 1 1 1 1 1 1
 1 1 1 1 ...
##  $ skewness_yaw_arm      : Factor w/ 395 levels "","-0.00311",..: 1 1 1 1 1 1
```

```
 1 1 1 1 ...
##  $ 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  : Factor w/ 398 levels "","-0.0035","-0.0073",..: 1
1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_dumbbell : Factor w/ 401 levels "","-0.0163","-0.0233",..: 1
1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_dumbbell   : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1
 1 ...
##  $ skewness_roll_dumbbell  : Factor w/ 401 levels "","-0.0082","-0.0096",..: 1
1 1 1 1 1 1 1 1 1 ...
##  $ skewness_pitch_dumbbell : Factor w/ 402 levels "","-0.0053","-0.0084",..: 1
1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_dumbbell   : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1
 1 ...
##  $ 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        : Factor w/ 73 levels "","-0.1","-0.2",..: 1 1 1 1 1
 1 1 1 1 1 ...
##  $ 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        : Factor w/ 73 levels "","-0.1","-0.2",..: 1 1 1 1 1
 1 1 1 1 1 ...
##  $ amplitude_roll_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##   [list output truncated]
```

```
sum( is.na( training ) )
```

```
## [1] 1287472
```

We notice that the data set contains many NA’s and missing values. Lets clean up some of the columns. Here we are only keeping columns with less than 500 missing values.

```
training <- Filter( function(x) (sum(x=="")<500), training)
sum( is.na( training ) )
```

```
## [1] 0
```

Finally remove the first 7 columns which are unrelated to the model fitting. Then divide up for training and validation sets.

```
training <- training[ ,-(1:7) ]
library( caret )
```

```
## Warning: package 'caret' was built under R version 3.1.3
```

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

```
## Warning: package 'lattice' was built under R version 3.1.3
```

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

```
## Warning: package 'ggplot2' was built under R version 3.1.3
```

```
set.seed( 97765 )
intrain <- createDataPartition( y=training$classe, p=0.7, list=FALSE )
trainsplit <- training[ intrain, ]
testsplit <- training[ -intrain, ]
```

Data Modelling

Here we will be using a Random Forest (RF) algorithm to train the data set on. Random Forrest are often one of the best machine learning algorithms in terms of their end accuracy. However, this comes at the cost of: high computational requirements (speed), the interpret ability of the final model and a tendency of the model to over fit onto the training data. To speed up the model processing time we will use parallel processing to divide the workload up onto multiple CPU cores. To reduce the chances of over fitting, the model will be using cross validation. Firstly we have already split the data up 70/30 into a training and testing data sets. This will allow us to check the model against a known data set. Further to that, the model will be using 10 fold cross validation during the learning process to reduce its variability.

```
library(parallel, quietly=T)
library(doParallel, quietly=T)
```

```
## Warning: package 'doParallel' was built under R version 3.1.3
```

```
## Warning: package 'foreach' was built under R version 3.1.3
```

```
## Warning: package 'iterators' was built under R version 3.1.3
```

```
## Turn on PP - With thanks to RAY JONES (TA)
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)

# 'classe' is the way in which the activcy was performed. Here we fit it against a
ll other vairables in the data set.
modelfit <- train( factor( trainsplit$classe ) ~ . , method="rf", data=trainsplit,
 trControl = trainControl(method="cv", number=10) )
```

```
## Loading required package: randomForest
```

```
## Warning: package 'randomForest' was built under R version 3.1.3
```

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

```
## Turn off PP
stopCluster(cluster)
modelfit
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 12362, 12363, 12362, 12364, 12363, 12364, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##    2    0.9919914  0.9898684  0.002305991  0.002917468
##   27    0.9912636  0.9889476  0.002119576  0.002682252
##   52    0.9852944  0.9813964  0.003199885  0.004047741
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

The model is reporting an accuracy of 99.2% Lets now compare our model performance against the data we split up for testing.

```
confmat1 <- confusionMatrix( trainsplit$classe, predict( modelfit ))
confmat2 <- confusionMatrix( testsplit$classe, predict( modelfit, testsplit ))
confmat2$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    0    0    0    0
##          B    9 1129    1    0    0
##          C    0    8 1018    0    0
##          D    0    0   18  945    1
##          E    0    0    0    2 1080
```

```
confmat2$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9933730      0.9916157      0.9909517      0.9952834      0.2859813
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

We can see that the model has performed very well on the testing set. The out of sample error rate is about 0.7%

Predicting

Now using the model fit, we can predict onto the testing set for answer submission.

```
answers <- as.character( predict( modelfit, testing ) )

# Code for exporting individual answer files for submission.
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)
```

The model scored 100%.

Conclusion

The goal of this project was to use use data from accelerators on participants performing barbell lifts to build a model able to predict the manner in which the exercise was performed. The model was built using a random trees algorithm with cross validation to identify the activity class.

The final model had an out of sample error of 0.7% on the testing data split. It correctly predicted all 20 cases of the test set of data.