Exercise Mode Prediction

localperf

Thursday, October 23, 2014

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, our goal is to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants to predict an exercise mode : either a correct exercise, or a wrong exercise execution in one of 4 ways. The 5 modes are coded in the data as A,B,C.D, and E.

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.

Variable Selection

```
library (caret)
```

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

```
library (dplyr)
```

```
##
## Attaching package: 'dplyr'
##
```

```
## The following object is masked from 'package:stats':
##
##     filter
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library (ggplot2)
library (randomForest)
```

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

```
get.data = function () {
   #--19622 rows, 160 columns
   #--three timestamp columns
   setwd ("d://coursera//machine//project")
   dir()
   train = read.csv ("pml-training.csv", na.strings = c("NA", "#DIV/0!", ""))
   train = tbl_df(train)
   colnames(train)[1] = "seq"
   train
   dim(train)
   summary(train)

   z = grep ("timestamp", colnames(train))
   colnames(train)[z]
   train = train[,-z]

   z = grep("user_name", colnames(train))
   train = train[,-z]

   train$seq         = NULL
   train$new_window  = NULL
   train$num_window  = NULL

   dim(train)
   head(colnames(train))
   train
}


explore = function (data) {
   #--count nas in each colname
   print (dim(data))
   tab = data.frame(var = colnames(data))
   for (var in colnames(data)) tab$nas[tab$var == var] = sum(is.na(data[,var]))
   tab = tab[order(tab$nas),]
   tab$seq = 1:dim(tab)[1]
   tab = tab[,c("seq", "var", "nas")]
   print (tab)
}
```

```
univariate = function (data, var) {
   #--plot sorted values of var, colored by class
   df                = data.frame(var = data[,var], classe = data$classe)
   df$classe         = as.character(df$classe)
   rownames(df)      = NULL
   colnames(df)[1]   = "var"
   df                = df[order(df$var, df$classe),]
   df$seq             = 1:dim(df)[1]
   runlengths        = rle(df$classe)
   lengths           = runlengths$lengths
   m                 = max(lengths)
   index.a           = which(lengths == m)
   index.b           = sum(lengths[1:(index.a-1)]) + 1
   index.c           = index.b + m - 1
   index.a
   index.b
   index.c
   x.ref       = c(index.b, index.c)
   msg.1       = paste("longest run =", max(runlengths$lengths))
   msg.2       = paste("number of runs =", length(runlengths$lengths))
   q           = qplot (x=seq, y = var, data = df, colour = df$classe, main = var )
   x.txt       =  .1 * dim(df)[1]
   y.txt       = min(df$var) + .8 * (max(df$var) - min(df$var))
   q           = q + geom_text(data = NULL, x = x.txt, y = y.txt,
                     label = paste(msg.1, msg.2, sep = "\n"))
   q           = q + geom_vline(xintercept = x.ref, colour = "red")
   #print (q)
   list (var = as.character(var), n = length(runlengths$lengths), max = max(runlengths$lengths),
         plot = q)
}


drop.na.columns = function (df) {
   #--drop columns if they have any NAs
   counts = data.frame(var = colnames(df))
   for (var in colnames(df)) {
      counts$na[counts$var == var] = sum (is.na(df[,var]))
   }
   head (counts)
  na.counts = colSums(is.na(df)) == 0
  table (na.counts)
  df2 = df[, na.counts ]
  df2
}


explore.univariate = function (df, preds) {
   stats = data.frame (var = preds)
   for (var in preds) {
      print                          (var)
      stat                           = univariate (df, var)
      print                          (stat)
```

```
      stats$runs[stats$var == var]    = stat$n
      stats$longest[stats$var == var] = stat$max
      stats$plot[stats$var == var]    = stat$plot
   }
   stats$var = as.character(stats$var)
   stats
}
```

‘‘‘

The training data have 19,622 rows and 160 columns. An examination of the first rows shows that the first column is a row sequence number, and the next six columns are user_name, three timestamp values, and two “window” variables. Since those seven fields are notphysiologic, they were deleted.

Further investidation showed that 100 columns were mostly “missing”: each had more than 19,000 missing values. Those columns were dropped, and the remaining data were saved as train2.

```
train = get.data()
explore (train) #--summarize columns by NA count
```

```
## [1] 19622   153
##     seq                       var   nas
## 1     1                roll_belt     0
## 2     2               pitch_belt     0
## 3     3                 yaw_belt     0
## 4     4         total_accel_belt     0
## 30    5             gyros_belt_x     0
## 31    6             gyros_belt_y     0
## 32    7             gyros_belt_z     0
## 33    8             accel_belt_x     0
## 34    9             accel_belt_y     0
## 35   10             accel_belt_z     0
## 36   11            magnet_belt_x     0
## 37   12            magnet_belt_y     0
## 38   13            magnet_belt_z     0
## 39   14                 roll_arm     0
## 40   15                pitch_arm     0
## 41   16                  yaw_arm     0
## 42   17          total_accel_arm     0
## 53   18              gyros_arm_x     0
## 54   19              gyros_arm_y     0
## 55   20              gyros_arm_z     0
## 56   21              accel_arm_x     0
## 57   22              accel_arm_y     0
## 58   23              accel_arm_z     0
## 59   24             magnet_arm_x     0
## 60   25             magnet_arm_y     0
## 61   26             magnet_arm_z     0
## 77   27            roll_dumbbell     0
## 78   28           pitch_dumbbell     0
## 79   29             yaw_dumbbell     0
## 95   30     total_accel_dumbbell     0
## 106  31         gyros_dumbbell_x     0
## 107  32         gyros_dumbbell_y     0
```

```
## 108  33         gyros_dumbbell_z     0
## 109  34         accel_dumbbell_x     0
## 110  35         accel_dumbbell_y     0
## 111  36         accel_dumbbell_z     0
## 112  37        magnet_dumbbell_x     0
## 113  38        magnet_dumbbell_y     0
## 114  39        magnet_dumbbell_z     0
## 115  40             roll_forearm     0
## 116  41            pitch_forearm     0
## 117  42              yaw_forearm     0
## 133  43      total_accel_forearm     0
## 144  44          gyros_forearm_x     0
## 145  45          gyros_forearm_y     0
## 146  46          gyros_forearm_z     0
## 147  47          accel_forearm_x     0
## 148  48          accel_forearm_y     0
## 149  49          accel_forearm_z     0
## 150  50         magnet_forearm_x     0
## 151  51         magnet_forearm_y     0
## 152  52         magnet_forearm_z     0
## 153  53                   classe     0
## 11   54            max_roll_belt 19216
## 12   55           max_picth_belt 19216
## 14   56            min_roll_belt 19216
## 15   57           min_pitch_belt 19216
## 17   58      amplitude_roll_belt 19216
## 18   59     amplitude_pitch_belt 19216
## 20   60     var_total_accel_belt 19216
## 21   61            avg_roll_belt 19216
## 22   62         stddev_roll_belt 19216
## 23   63            var_roll_belt 19216
## 24   64           avg_pitch_belt 19216
## 25   65        stddev_pitch_belt 19216
## 26   66           var_pitch_belt 19216
## 27   67             avg_yaw_belt 19216
## 28   68           stddev_yaw_belt 19216
## 29   69              var_yaw_belt 19216
## 43   70            var_accel_arm 19216
## 44   71             avg_roll_arm 19216
## 45   72          stddev_roll_arm 19216
## 46   73             var_roll_arm 19216
## 47   74            avg_pitch_arm 19216
## 48   75         stddev_pitch_arm 19216
## 49   76            var_pitch_arm 19216
## 50   77              avg_yaw_arm 19216
## 51   78           stddev_yaw_arm 19216
## 52   79              var_yaw_arm 19216
## 68   80             max_roll_arm 19216
## 69   81            max_picth_arm 19216
## 70   82              max_yaw_arm 19216
## 71   83             min_roll_arm 19216
## 72   84            min_pitch_arm 19216
## 73   85              min_yaw_arm 19216
## 74   86       amplitude_roll_arm 19216
```

```
## 75   87      amplitude_pitch_arm 19216
## 76   88        amplitude_yaw_arm 19216
## 86   89        max_roll_dumbbell 19216
## 87   90       max_picth_dumbbell 19216
## 89   91        min_roll_dumbbell 19216
## 90   92       min_pitch_dumbbell 19216
## 92   93  amplitude_roll_dumbbell 19216
## 93   94 amplitude_pitch_dumbbell 19216
## 96   95       var_accel_dumbbell 19216
## 97   96        avg_roll_dumbbell 19216
## 98   97     stddev_roll_dumbbell 19216
## 99   98        var_roll_dumbbell 19216
## 100  99       avg_pitch_dumbbell 19216
## 101 100    stddev_pitch_dumbbell 19216
## 102 101       var_pitch_dumbbell 19216
## 103 102         avg_yaw_dumbbell 19216
## 104 103      stddev_yaw_dumbbell 19216
## 105 104         var_yaw_dumbbell 19216
## 124 105         max_roll_forearm 19216
## 125 106        max_picth_forearm 19216
## 127 107         min_roll_forearm 19216
## 128 108        min_pitch_forearm 19216
## 130 109   amplitude_roll_forearm 19216
## 131 110  amplitude_pitch_forearm 19216
## 134 111        var_accel_forearm 19216
## 135 112         avg_roll_forearm 19216
## 136 113      stddev_roll_forearm 19216
## 137 114         var_roll_forearm 19216
## 138 115        avg_pitch_forearm 19216
## 139 116     stddev_pitch_forearm 19216
## 140 117        var_pitch_forearm 19216
## 141 118          avg_yaw_forearm 19216
## 142 119       stddev_yaw_forearm 19216
## 143 120          var_yaw_forearm 19216
## 84  121  skewness_pitch_dumbbell 19217
## 81  122  kurtosis_picth_dumbbell 19218
## 83  123   skewness_roll_dumbbell 19220
## 80  124   kurtosis_roll_dumbbell 19221
## 88  125         max_yaw_dumbbell 19221
## 91  126         min_yaw_dumbbell 19221
## 94  127   amplitude_yaw_dumbbell 19221
## 8   128       skewness_roll_belt 19225
## 5   129       kurtosis_roll_belt 19226
## 13  130             max_yaw_belt 19226
## 16  131             min_yaw_belt 19226
## 19  132       amplitude_yaw_belt 19226
## 64  133         kurtosis_yaw_arm 19227
## 67  134         skewness_yaw_arm 19227
## 6   135      kurtosis_picth_belt 19248
## 9   136     skewness_roll_belt.1 19248
## 65  137        skewness_roll_arm 19293
## 62  138        kurtosis_roll_arm 19294
## 63  139       kurtosis_picth_arm 19296
## 66  140       skewness_pitch_arm 19296
```

```
## 121 141    skewness_roll_forearm 19299
## 118 142    kurtosis_roll_forearm 19300
## 126 143          max_yaw_forearm 19300
## 129 144          min_yaw_forearm 19300
## 132 145    amplitude_yaw_forearm 19300
## 119 146   kurtosis_picth_forearm 19301
## 122 147   skewness_pitch_forearm 19301
## 7   148        kurtosis_yaw_belt 19622
## 10  149        skewness_yaw_belt 19622
## 82  150    kurtosis_yaw_dumbbell 19622
## 85  151    skewness_yaw_dumbbell 19622
## 120 152     kurtosis_yaw_forearm 19622
## 123 153     skewness_yaw_forearm 19622
```

```
train2 = drop.na.columns(train)
dim(train2)
```

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

At this point, train2 has 52 predictors, and one one outcome ("classe").

Random Forest

I decided to use the randomForset procedure in the R randomForest package.

```
set.seed (1188)
ntree = 50
fit.rf = randomForest (classe ~ ., data = train2, importance = T, ntree = ntree)
predicted = predict (fit.rf, train, type = "class")
mc = table (train2$classe, predicted)
mc
```

```
##    predicted
##        A    B    C    D    E
##   A 5580    0    0    0    0
##   B    0 3797    0    0    0
##   C    0    0 3422    0    0
##   D    0    0    0 3216    0
##   E    0    0    0    0 3607
```

```
summary(fit.rf)
```

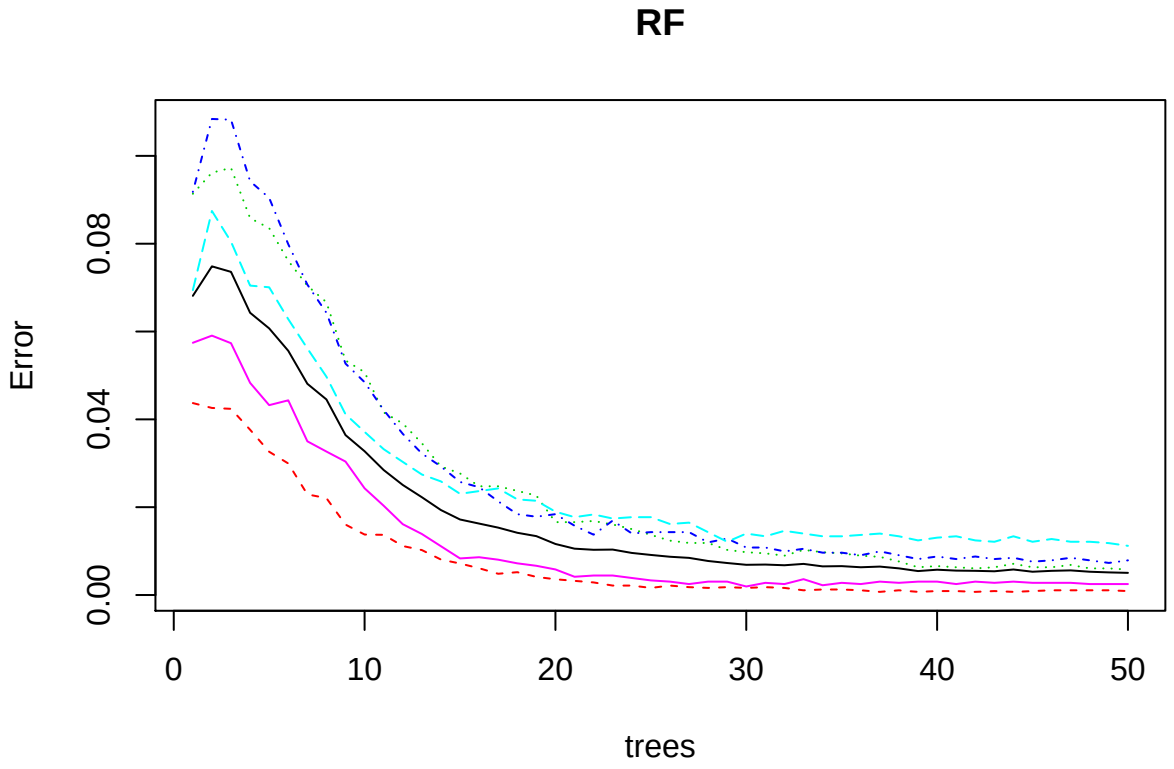
```
##                 Length Class  Mode
## call               5  -none- call
## type               1  -none- character
## predicted      19622  factor numeric
## err.rate         300  -none- numeric
## confusion         30  -none- numeric
## votes          98110  matrix numeric
## oob.times      19622  -none- numeric
```

```
## classes             5  -none- character
## importance        364  -none- numeric
## importanceSD      312  -none- numeric
## localImportance     0  -none- NULL
## proximity           0  -none- NULL
## ntree               1  -none- numeric
## mtry                1  -none- numeric
## forest             14  -none- list
## y               19622  factor numeric
## test                0  -none- NULL
## inbag               0  -none- NULL
## terms               3  terms  call
```

```
confusionMatrix (train2$classe, predicted)
```

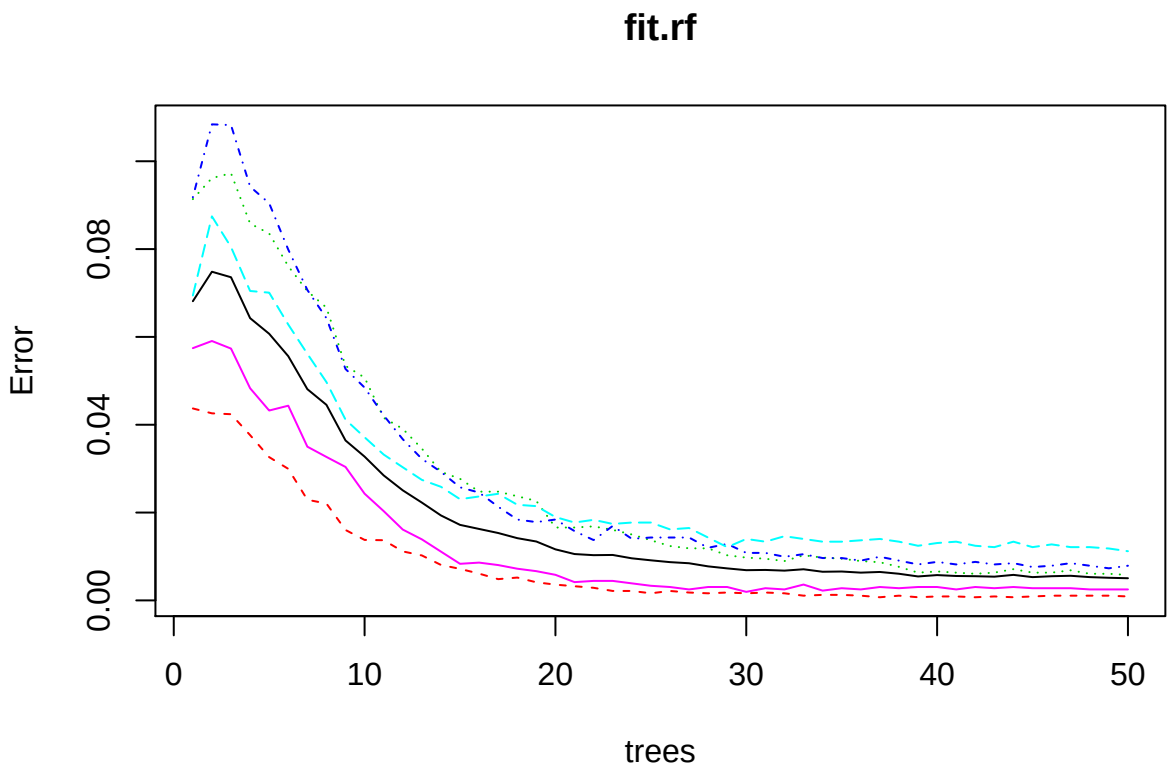
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 5580    0    0    0    0
##          B    0 3797    0    0    0
##          C    0    0 3422    0    0
##          D    0    0    0 3216    0
##          E    0    0    0    0 3607
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (1, 1)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             1.000    1.000    1.000    1.000    1.000
## Specificity             1.000    1.000    1.000    1.000    1.000
## Pos Pred Value          1.000    1.000    1.000    1.000    1.000
## Neg Pred Value          1.000    1.000    1.000    1.000    1.000
## Prevalence              0.284    0.194    0.174    0.164    0.184
## Detection Rate          0.284    0.194    0.174    0.164    0.184
## Detection Prevalence    0.284    0.194    0.174    0.164    0.184
## Balanced Accuracy       1.000    1.000    1.000    1.000    1.000
```

```
plot (fit.rf, main = "RF")
```



The accuracy of the classifier is 100% - an unexpected outcome. The ntree parameter in the call to randomForest specifies a number of trees to be built, and deafults to 500. I reduced the value until the accuracy was less than 100%, which occurred near 15 trees. The plot below shows that mis-classification rates fall off very rapidly with ibncreasing tree counts.

```
plot (fit.rf)
```



Cross-validation

I decided to implement 10-fold cross validation.

To do this, I divided the training data into 10 nearly-equally-sized folds, with a random assigment of rows to folds. For each fold, I created a test and training set coposed of all the training data but the fold, and the fold, respectively. I used the same call to randomForest 10 times, once with each of the new training subset; and then scored that model on the held out data.

```
k.fold.rf.cross = function (df, ntree, K = 10) {
   #--build K models, holding out 1/K of the data each time
   #--and then scoring against the held out data
   n  = dim (df)[1]
   u  = runif (n)
   df = df[order(u),]
   length(unique(u))   #--shuffle the rows

   row = 1:dim(df)[1]
   folds = row %% K
   head (folds, 20)
   accuracy = NULL
   table (folds)
   for (fold in unique(folds)) {
      local = df[-which(folds == fold),]
      test  = df[ which(folds == fold),]
```

```
      local.fit = randomForest (classe ~ ., data = local, ntree=ntree, type = "class")
      predicted = predict (local.fit, test, type = "class")
      mc = table (test$classe, predicted)
      accuracy = c(accuracy, sum(diag(mc)) / sum (mc))
   }
   accuracy
}

set.seed (271828)
rm (accuracy)
```

```
## Warning: object 'accuracy' not found
```

```
accuracy = k.fold.rf.cross (train2, ntree=ntree, K = 10)
summary(accuracy)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.993   0.996   0.996   0.996   0.996   0.997
```

As shown above, the accuracy remains very high - above 99% for all 10 folds.

Variable Importance

The Gini importance of each variable in contributing to model accuracy is saved in the fit object returned by randomForest. The 5 most important and 5 least important variables are shown below:

```
imp.rf = as.data.frame(randomForest::importance(fit.rf, type = 2))
imp.rf$var = row.names(imp.rf)
imp.rf = imp.rf[order(-imp.rf$MeanDecreaseGini),]
head (imp.rf)
```

```
##                   MeanDecreaseGini              var
## roll_belt                   1083.9         roll_belt
## yaw_belt                    1005.5          yaw_belt
## magnet_dumbbell_z            789.5 magnet_dumbbell_z
## pitch_forearm                738.2     pitch_forearm
## pitch_belt                   738.0        pitch_belt
## magnet_dumbbell_y            649.0 magnet_dumbbell_y
```

```
tail (imp.rf)
```

```
##                  MeanDecreaseGini             var
## gyros_belt_y                99.11     gyros_belt_y
## gyros_belt_x                96.18     gyros_belt_x
## gyros_forearm_z             84.61  gyros_forearm_z
## gyros_dumbbell_z            82.58 gyros_dumbbell_z
## gyros_forearm_x             76.30  gyros_forearm_x
## gyros_arm_z                 50.71      gyros_arm_z
```

Scoring the Test Data

(TBD)