Human Activity Recognition

R. Srivastav

Saturday, November 22, 2014

Abstract We examined the data analyzed by Ugulino et al. in Qualitative Activity Recognition of Weight Lifting Exercises and attempted to match their performance in detecting the correct execution of a Unilateral Dumbbell Biceps Curls (class A) and classifying common errors such as throwing the elbows to the front (class B) or lowering the dumbbell only halfway (class D).

Our goal was to develop an accurate parsimonious model which came close to matching the 98.03% overall recognition performance they report.

Several different techniques were examined but ultimately we found that using random forests on summary variables yielded excellent reliability.

Data Processing The code below shows how we modified the raw data. Briefly, we eliminated missing values, and also removed timestamps and information identifying the individual who was performing the exercises. Both might be useful in developing a more accurate, personalized model for guiding an exercise regime. In addition, we also decided not to look at measurements along the three dimensions, opting instead to analyze just a summary statistic — roll-belt instead of roll-belt-x, roll-belt-y, and roll-belt-z.

The data for this study came from http://groupware.les.inf.puc-rio.br/har.

```r
train_url <- 'http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv'
test_url  <- 'http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv'

train_file <- './pml-training.csv'
test_file  <- './pml-testing.csv'

force <- 0 # helps control reruns of file reads
if (force | ! file.exists(train_file)) {
    download.file(train_url, destfile = train_file)
}

if (force | ! file.exists(test_file)) {
    download.file(test_url, destfile = test_file)
}

raw_train <- read.csv(train_file,na.strings=c("", "NA","#DIV/0!")) ## YAW
raw_test  <- read.csv(test_file,na.strings=c("", "NA","#DIV/0!"))

# only use columns which are not missing data - see how well this works?

use <- colSums(is.na(raw_train))==0
proc_data <- raw_train[,use]
proc_test <- raw_test[,use]

# remove also columns corresponding to id and summary info. Incorporate timestamps diffs later

dontuse <- grep('X|user|timestamp|window|total',names(proc_data))
proc_data <- proc_data[,-dontuse]
proc_test <- proc_test[,-dontuse]
```

```
dontuse <- grep('_x|_y|_z',names(proc_data))
proc_data <- proc_data[,-dontuse]
proc_test <- proc_test[,-dontuse]

set.seed(121)

library(caret); library(lattice); library(ggplot2)
```

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

```
# create test and validation sets
# note that test file above is not the same as the testing file below which is used for cross-validatio

use <- createDataPartition(y=proc_data$classe, p=0.8, list=FALSE)
training <- proc_data[ use,]
testing  <- proc_data[-use,]
```

Sample Data (a peek) The data was partitioned into a test and validation set. The tables below show the number of observations sorted by class and summary statistics for a few important variables. There were four common errors which were classified in addition to the correct execution (class A).

```
summary(training[,c(13,1,6,9,11)])
```

```
##  classe     roll_belt         yaw_arm           yaw_dumbbell
##  A:4464   Min.   :-28.90   Min.   :-180.000   Min.   :-150.871
##  B:3038   1st Qu.:  1.09   1st Qu.: -43.500   1st Qu.: -77.733
##  C:2738   Median :113.00   Median :   0.000   Median :  -3.660
##  D:2573   Mean   : 64.41   Mean   :  -0.839   Mean   :   1.608
##  E:2886   3rd Qu.:123.00   3rd Qu.:  45.650   3rd Qu.:  79.505
##           Max.   :162.00   Max.   : 180.000   Max.   : 154.952
##  pitch_forearm
##  Min.   :-72.50
##  1st Qu.:  0.00
##  Median :  8.97
##  Mean   : 10.55
##  3rd Qu.: 28.20
##  Max.   : 88.70
```

Results As noted above we looked at many different predictive models and also many different predictor sets, but the simplest also seemed to be the best. Ultimately, we used a random forest with just 12 features. The forest size was set to four although using even just two, yields only marginally lower predictive accuracy.

```
ctrl <- trainControl(method = "cv", number = 4, allowParallel = TRUE)
fit <- train(classe ~., method="rf", data=training, trControl=ctrl)
```

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

```r
fit
```

```
## Random Forest
##
## 15699 samples
##    12 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
##
## Summary of sample sizes: 11774, 11775, 11774, 11774
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##    2    0.9833747  0.9789715  0.001028825  0.001299937
##    7    0.9836932  0.9793745  0.001082316  0.001367455
##   12    0.9779603  0.9721273  0.001464772  0.001847573
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 7.
```

The table below shows that the classifier was nearly perfect for the training set.

```r
confusionMatrix(predict(fit,training),training$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 4464    0    0    0    0
##          B    0 3038    0    0    0
##          C    0    0 2738    0    0
##          D    0    0    0 2573    0
##          E    0    0    0    0 2886
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9998, 1)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Prevalence   0.2843   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

The results for the validation set were nearly as good. Note that validation is strictly speaking not necessary for this algorithm, but it does provide a convenient metric for judging how well we did.

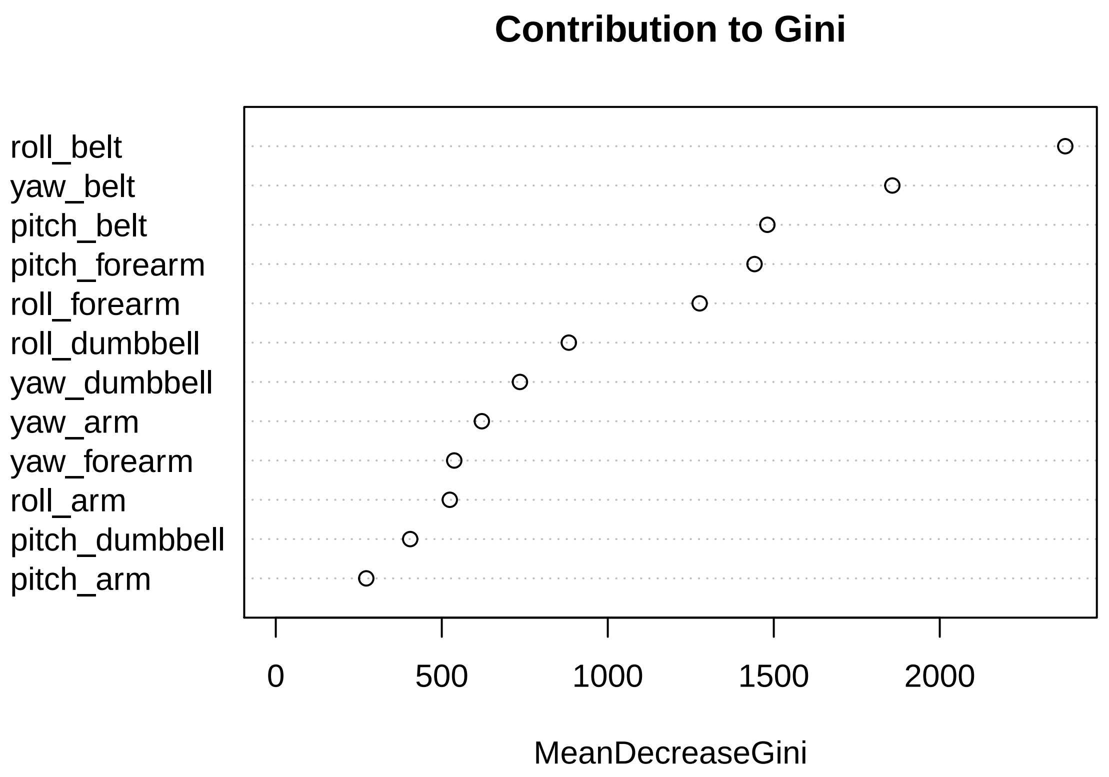
```
conf <- confusionMatrix(predict(fit,testing), testing$classe)
conf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1110    8    0    0    0
##          B    3  742    7    1    3
##          C    0    9  670    7    1
##          D    1    0    6  635    3
##          E    2    0    1    0  714
##
## Overall Statistics
##
##                Accuracy : 0.9867
##                  95% CI : (0.9827, 0.9901)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9832
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9946   0.9776   0.9795   0.9876   0.9903
## Specificity            0.9971   0.9956   0.9948   0.9970   0.9991
## Pos Pred Value         0.9928   0.9815   0.9753   0.9845   0.9958
## Neg Pred Value         0.9979   0.9946   0.9957   0.9976   0.9978
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2829   0.1891   0.1708   0.1619   0.1820
## Detection Prevalence   0.2850   0.1927   0.1751   0.1644   0.1828
## Balanced Accuracy      0.9959   0.9866   0.9871   0.9923   0.9947
```

We feel comfortable reporting a predictive accuracy between 98.3 and 99 percent. Ugulino et al. obtained an accuracy over 99% using more variables and ten trees.

The graph below shows how much each variable contributes to the quality of classification. With minor differences, results are robust over mutlple runs of the random forest algorithm.

```
varImpPlot(fit$finalModel,main="Contribution to Gini")
```



Conclusion The classifier derived here performs well. We are especially pleased that it has the highest balanced accuracy and excellent positive and negative predictive values — meaning that it rarely transmits incorrect feedback to the athelete. Training with a minimal number of trees and a smaller feature set may or may not be an asset in the deployment of the tool depending on the degree of personalization desired.