Human Activity Recognition Predictive Model

D Ritchie

May 21, 2016

Human Activity Recognition

Human Activity Recognition - HAR - has emerged as a key research area in the last years and is gaining increasing attention by the pervasive computing research community, especially for the development of context-aware systems. There are many potential applications for HAR, like: elderly monitoring, life log systems for monitoring energy expenditure and for supporting weight-loss programs, and digital assistants for weight lifting exercises. The data for this project come from this source: http://groupware.les.inf.puc-rio.br/har. They have been very generous in allowing their data to be used for this kind of assignment.

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.

Weight Lifting Exercises Dataset

Human activity recognition research has traditionally focused on discriminating between different activities, i.e. to predict "which" activity was performed at a specific point in time. The approach they proposed for the Weight Lifting Exercises dataset was to investigate "how (well)" an activity was performed by the wearer.

In this work they first define quality of execution and investigate three aspects that pertain to qualitative activity recognition: the problem of specifying correct execution, the automatic and robust detection of execution mistakes, and how to provide feedback on the quality of execution to the user. They tried out an on-body sensing approach, but also an "ambient sensing approach" (by using Microsoft Kinect - dataset still unavailable).

Six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions: exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E).

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).

Project Scope

In this project, my goal is to use data from accelerometers on the belt, forearm, arm, and dumbbell of 6 participants and accurately predict if they performed correctly or incorrectly based on the "classe" variable. I will use the Caret, Random Forest and GMB packages within R.

Results and Conclusion

Since this is a classification prediction, I used Random Forest (RF) for my final model and ran a Generalized Boosted Model (GBM) for comparison based on my system resources and their ability to address 'empty' data fields within themselves. Both were run using K-Fold cross validation of 5 (again system resources and time were a factor), which allows repeating the cross validation function with exactly the same splitting results for each repetition. I used 70% of the 19,622 observations available for my training set.

Random Forest produced an average accuracy of 98.49% and the Generalized Boosted Model had accuracy of 92.59%. I used the RF model output on the test data of 20 observations and was able to predict with 100% accuracy. Both produce promising Confusion Matrix's and I found it very interesting in how the different models top 15 variants of importance differed. (see Model Runs below)

Package Install and Data Extraction

install.packages("caret"), library(caret) require(caret)

install.packages("randomForest") require(randomForest)

```
testData=read.csv("~/pmltesting.csv", header = TRUE)
dim(testData)

## [1]  20 160

#[1]  20 160

#clean up useless columns 1-8
testData<-testData[,-c(1:8)]
dim(testData)

## [1]  20 152

#20 152

trainData=read.csv("~/pmltraining.csv", header = TRUE, na.strings = c("NA",
"", "#DIV/0"))
dim(trainData)

## [1] 19622   160

#[1] 19622   160
```

Data Review and Cleaning

In reviewing the data I found multiple variables with empty data on 19,622 observations. Although there are ways to 'fill in' missing data I chose to excluded them. I was able to reduce my variants from 160 to 52. Earlier attempts to used all variables in model selections were overwhelming my systems capacity. I felt 52 variables to be an acceptable amount based on classifying 5 types. Had there been 40+ class types, I might have found other options of addressing the empty data fields.

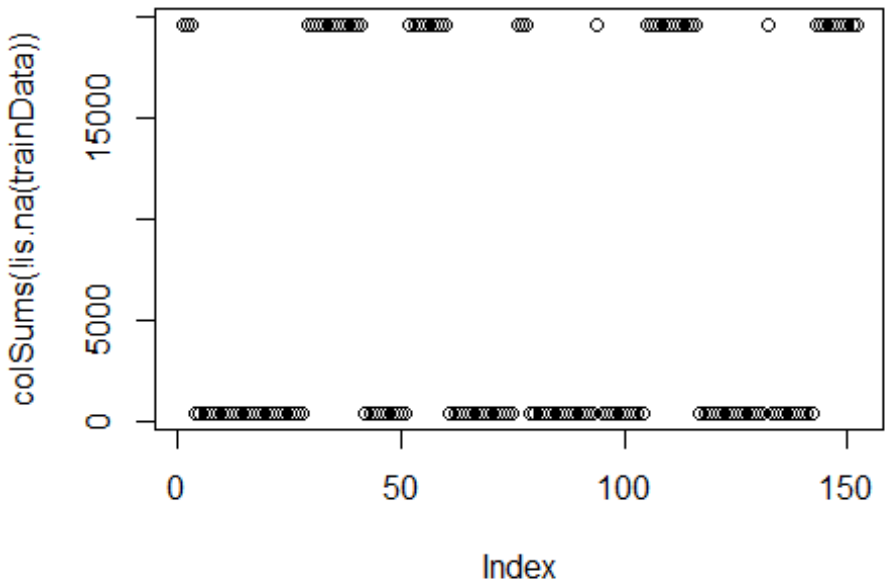
Let go of the few easily identified as not needed.

```
trainData<-trainData[,-c(1:8)]
dim(trainData)

## [1] 19622   152

#[1] 19622   152
```

A quick look at the variables to see how many are missing data.

I used colSums(!is.na(trainData)) to view a count in all 152 remaining variables. Two numbers prevalent when scrolling through the information, 406 and 19,622. A quick plot of this supports my finding, but doesn't easily identify which columns.

```
plot(colSums(!is.na(trainData)))
```



After several iterations (not shown here) I come to the conclusion that manually identifying and clearing variables isn't very practical, nor could I justify removal of columns by a percentage of my choosing. Why not 20% or 50% or 80%? The plot appears to show a

sizable amount of variables having data, so my approach is to remove all columns with missing data.

```
#NA and open data cleaning on varialbes
trainData<-trainData[, apply(trainData, 2, function (x) !any(is.na(x)))]
dim(trainData)

## [1] 19622    52

#19622    52

library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

InTrain<-createDataPartition(trainData$classe, p=0.70, list=FALSE)
training1<-trainData[InTrain,]
testing1=trainData[-InTrain, ]
```

Model Runs

I now present, a running of the bulls. After many late nights on this I envisioned these trees and project, as more like a "running of the bulls", where I was better off getting out of the way and using the model defaults where possible.

Random Forest

```
library(randomForest)

## randomForest 4.6-12

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

set.seed(711)

fitControl<-trainControl(method="cv", number=5, allowParallel=T, verbose=T,
classProbs = T)
RFfit<-train(classe~.,data=training1, method="rf", trControl=fitControl,
verbose=F)

## + Fold1: mtry= 2
## - Fold1: mtry= 2
## + Fold1: mtry=26
## - Fold1: mtry=26
```

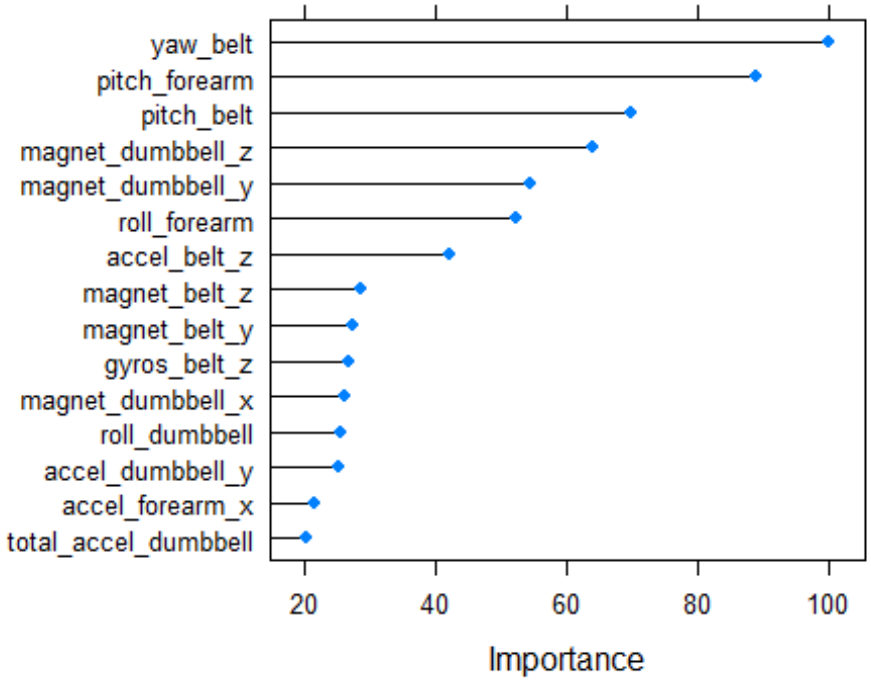
```
## + Fold1: mtry=51
## - Fold1: mtry=51
## + Fold2: mtry= 2
## - Fold2: mtry= 2
## + Fold2: mtry=26
## - Fold2: mtry=26
## + Fold2: mtry=51
## - Fold2: mtry=51
## + Fold3: mtry= 2
## - Fold3: mtry= 2
## + Fold3: mtry=26
## - Fold3: mtry=26
## + Fold3: mtry=51
## - Fold3: mtry=51
## + Fold4: mtry= 2
## - Fold4: mtry= 2
## + Fold4: mtry=26
## - Fold4: mtry=26
## + Fold4: mtry=51
## - Fold4: mtry=51
## + Fold5: mtry= 2
## - Fold5: mtry= 2
## + Fold5: mtry=26
## - Fold5: mtry=26
## + Fold5: mtry=51
## - Fold5: mtry=51
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 26 on full training set
```

```
print(RFfit)
```

```
## Random Forest
##
## 13737 samples
##    51 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10990, 10989, 10991, 10989, 10989
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9888622  0.9859092
##   26    0.9911189  0.9887645
##   51    0.9850040  0.9810277
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 26.
```

```
print(RFfit$finalModel)
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, verbose = ..1)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 26
##
##         OOB estimate of  error rate: 0.67%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3899    5    1    0    1 0.001792115
## B   14 2632   12    0    0 0.009781791
## C    0   13 2378    5    0 0.007512521
## D    0    0   31 2220    1 0.014209591
## E    0    0    2    7 2516 0.003564356
```



Test results using RF as final model.

```
predRF<-predict(RFfit, newdata=testData)
print(predRF)
```

```
##  [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]  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
```

Generalized Boosted Model

```
library(gbm)
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.1
```

```
set.seed(711)
```

```
GBMfit<-train(classe~., data=training1, method="gbm", trControl=fitControl,
verbose=F)
```

```
## Loading required package: plyr
```

```
## + Fold1: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10,
n.trees=150
## - Fold1: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10,
n.trees=150
## + Fold1: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10,
n.trees=150
## - Fold1: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10,
n.trees=150
## + Fold1: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10,
n.trees=150
## - Fold1: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10,
n.trees=150
## + Fold2: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10,
n.trees=150
## - Fold2: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10,
n.trees=150
## + Fold2: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10,
n.trees=150
## - Fold2: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10,
n.trees=150
## + Fold2: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10,
n.trees=150
## - Fold2: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10,
n.trees=150
## + Fold3: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10,
n.trees=150
```

```
## - Fold3: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10,
n.trees=150
## + Fold3: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10,
n.trees=150
## - Fold3: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10,
n.trees=150
## + Fold3: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10,
n.trees=150
## - Fold3: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10,
n.trees=150
## + Fold4: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10,
n.trees=150
## - Fold4: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10,
n.trees=150
## + Fold4: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10,
n.trees=150
## - Fold4: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10,
n.trees=150
## + Fold4: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10,
n.trees=150
## - Fold4: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10,
n.trees=150
## + Fold5: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10,
n.trees=150
## - Fold5: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10,
n.trees=150
## + Fold5: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10,
n.trees=150
## - Fold5: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10,
n.trees=150
## + Fold5: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10,
n.trees=150
## - Fold5: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10,
n.trees=150
## Aggregating results
## Selecting tuning parameters
## Fitting n.trees = 150, interaction.depth = 3, shrinkage = 0.1,
n.minobsinnode = 10 on full training set
```

```
print(GBMfit)
```

```
## Stochastic Gradient Boosting
##
## 13737 samples
##    51 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10990, 10989, 10991, 10989, 10989
```

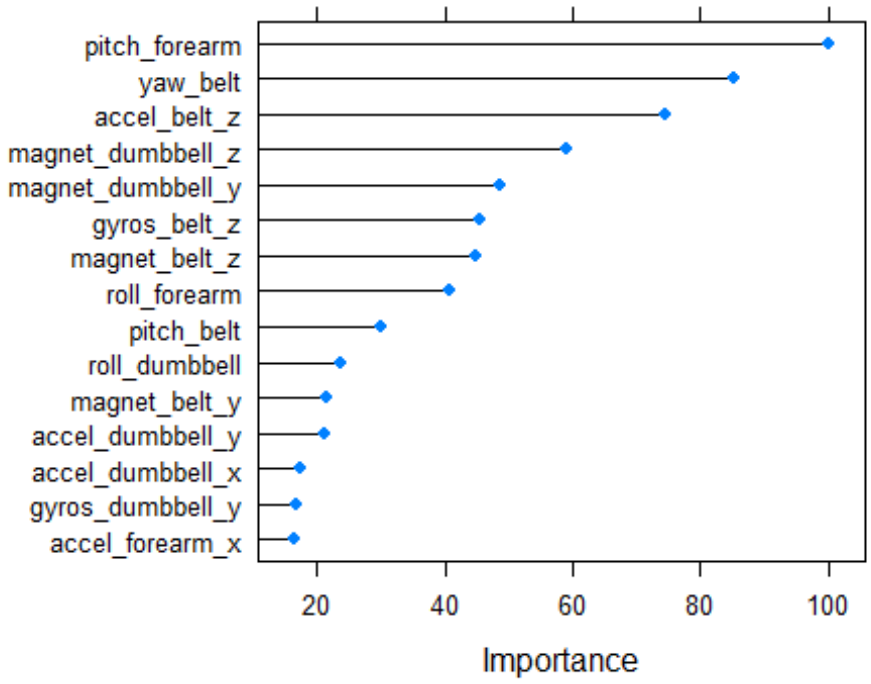
```
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7423763  0.6732374
##   1                  100      0.8123324  0.7624333
##   1                  150      0.8456728  0.8046776
##   2                   50      0.8544081  0.8154556
##   2                  100      0.9065299  0.8817080
##   2                  150      0.9312809  0.9130366
##   3                   50      0.8929168  0.8644255
##   3                  100      0.9418358  0.9263985
##   3                  150      0.9590887  0.9482398
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
print(GBMfit$finalModel)
```

```
## A gradient boosted model with multinomial loss function.
## 150 iterations were performed.
## There were 51 predictors of which 45 had non-zero influence.
```

```
print(confusionMatrix(GBMfit))
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    A    B    C    D    E
##          A 28.0  0.7  0.0  0.0  0.0
##          B  0.3 18.0  0.6  0.1  0.2
##          C  0.1  0.6 16.6  0.6  0.2
##          D  0.0  0.0  0.2 15.6  0.2
##          E  0.0  0.0  0.0  0.2 17.7
##
##  Accuracy (average) : 0.9591
```




```
predGBM<-predict(GBMfit, newdata = testData)
print(predGBM)

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

Conclusion

In conclusion, the RF model produced an OOB estimate of error rate at 0.67% on a sample size of ~10,989 out of possible 13,737 observations using 500 trees with 2 variables at each split and cross validated resampling of 5 fold. The final model value of mtry = 2 resulted in a 99.11% accuracy and 98.87% kappa. When tested against the testing model, 100% accuracy was obtained.

The GBM model held the shrinkage tuning parameter constant at 0.1 and the n.nimodsinnode at 10. The final model values used were ntrees = 150, an interaction depth = 3 which produced an average accuracy of 95.51% and a kappa of 94.82%. The GBM final model produce similar results on the test data, with the exception of observation 11 where RF returned a value of B and GBM returned a value of C. Overall the GBM is a acceptable tool missing 1 out of 20 for a 5% error rate.

In running these models numerous times, I've found that almost always the resulting testing outputs to be exactly the same. I also found it interesting reviewing the different models variances of importance and how they differ. My overall findings and growth from

performing this exercise had helped me grow from further development of using the Applied Predictive Modeling book in PDF form, which was recommended during course lectures. I highly recommend it to everyone. The PDF version allowed for quicker subject searches and cross referencing. The http://topepo.github.io/caret/index.html recommended in lecture was also a good resource. Also, I encountered system constraints during this process and found using ReadyBoost in windows 7 as a valuable and cheap approach to increasing my cache size related to my operating system. It uses usb drive flash memory, which everyone should have. Note this my not be available if you are working off a hard disc.