Practical Machine Learning - Course Project

Akira Sano

4/2/2020

Introduction

The 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 (like with the Daily Living Activities dataset above). The approach the investigators propose for the Weight Lifting Exercises dataset is to investigate "how (well)" an activity was performed by the wearer. The "how (well)" investigation has only received little attention so far, even though it potentially provides useful information for a large variety of applications,such as sports training.

In this work, the investigators 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. We tried out an on-body sensing approach (dataset here), 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). Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes.

One goal is to apply machine learning techniques to the given accelerometer data to predict whether an observation (a human), with various features recordable in the data, executes the lifting correctly (class A) or not (other classes).

Data Collection and Cleaning

We download the training and testing data sets and partition the training set into the training (70%) and testing sets (30%). The testing data contains 20 observations and labeled as a validating set. The empty and #DIV/0! entries convert to NAs.

```
#setwd("Coursera/Machine Learning/")
#download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv", "pml-training.cs
dat <- read.csv("pml-training.csv", na.strings = c("NA", "", "#DIV/0!"))
#download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv", "pml-testing.csv"
validating <- read.csv("pml-testing.csv", na.strings = c("NA", "", "#DIV/0!"))
```

The variable "classe" is the class variable.

```
table(dat$classe)
```

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

We will remove unnecessary columns from the set dat, for prediction such as index (X), and do some data cleaning (remove columns with very few observations), then partition it into training and testing set.

```
#Remove "X" column (row index)
dat <- subset(dat, select = -X)
validating <- subset(validating, select = -X)
```

At this point, every row (observation) has NA in at least one of its columns.

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.6.3
```

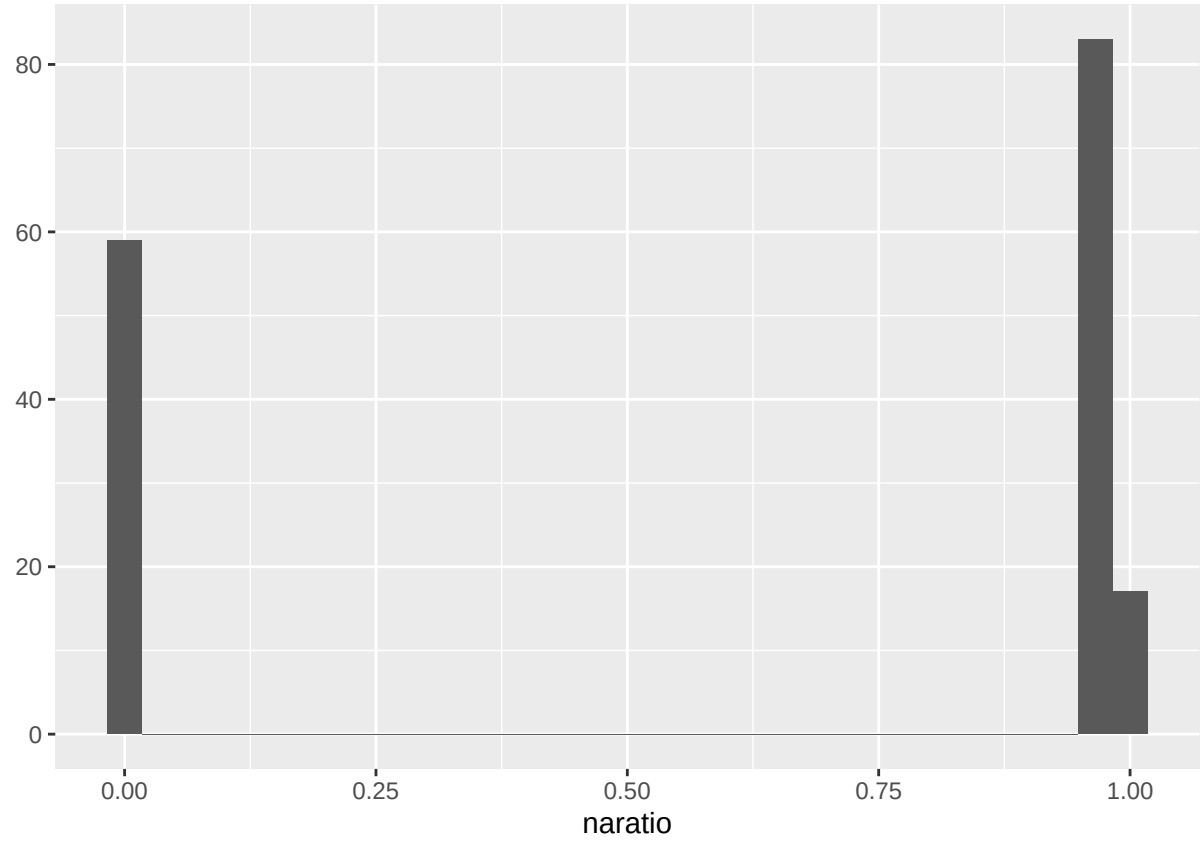
```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 3.6.3
```

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

```
naratio <- sapply( dat, function(x) sum(is.na(x))/dim(dat)[1] )
qplot(naratio)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We remove all the columns whose NA rate is greater than 80%.

```
dat <- subset(dat, select = names(naratio)[naratio < 0.8])
validating <- subset(validating, select = names(naratio)[naratio < 0.8][-59]) # col 59 = classe which i
```

Further, remove near-zero-variance variablesfrom dat and align the validating set to it.

```
tmp <- nearZeroVar(dat, saveMetrics = TRUE)
dat <- subset(dat,  select = rownames(tmp[tmp$nzv == FALSE,]))
validating <- subset(validating,  select = rownames(tmp[tmp$nzv == FALSE,])[-58]) #col 58 = classe
```

raw_timestamp_part_1 is equal to cvtd_timestamp in seconds. We find the correct conversion (and remove cvtd_timestamp) *but do not use it*, and keep the microseconds in raw_timestamp_part_2 as seconds.

```
#dat$datetime <- as.POSIXct(dat$raw_timestamp_part_1, "%d/%m/%Y %H:%M", origin="1970-01-01")
#validating$datetime <- as.POSIXct(validating$raw_timestamp_part_1, "%d/%m/%Y %H:%M", origin="1970-01-0
dat$raw_timestamp_part_1 <- NULL
validating$raw_timestamp_part_1 <- NULL
dat$dur <- dat$raw_timestamp_part_2 / 1.0e6
validating$dur <- validating$raw_timestamp_part_2 / 1.0e6
dat$raw_timestamp_part_2 <- NULL
validating$raw_timestamp_part_2 <- NULL

dat <- data.frame(dat[,-56], classe=dat[,56])
dat$cvtd_timestamp <- NULL
validating$cvtd_timestamp <- NULL
```

We now partition dat into the training and testing sets (reproducible).

```
library(caret)
set.seed(123)
inTrain <- createDataPartition(dat$classe, p = 0.7, list = FALSE)
training <- dat[inTrain,]
testing <- dat[-inTrain,]
```

Model fits

We use randomForest and gbm packages to train and predict on the test set by cross validating.

```
ctl <- trainControl(method = "cv", number = 4, allowParallel = TRUE)

fitrf <- train(classe ~., method = "rf", trControl = ctl, data = training)
fitgbm <- train(classe ~., method = "gbm", trControl = ctl, data = training, verbose = FALSE)
```

```
prf <- predict(fitrf, testing)
pgbm <- predict(fitgbm, testing)
table(prf, testing$classe)
```

```
##
## prf    A    B    C    D    E
```

```
##   A 1674    2    0    0    0
##   B    0 1134    1    0    0
##   C    0    3 1025    1    0
##   D    0    0    0  963    3
##   E    0    0    0    0 1079
```

```
table(pgbm, testing$ classe)
```

```
##
## pgbm    A    B    C    D    E
##    A 1670    5    0    0    2
##    B    3 1117   11   10    0
##    C    0   16 1013    8    5
##    D    0    1    2  946   12
##    E    1    0    0    0 1063
```

```
table(prf, pgbm)
```

```
##    pgbm
## prf    A    B    C    D    E
##   A 1671    4    0    0    1
##   B    4 1116   14    1    0
##   C    0   11 1016    2    0
##   D    0   10    7  949    0
##   E    2    0    5    9 1063
```

In random forest, if we use the "importance" function, it is possible to glance at which variables are important (in the sense of accuracy drop in prediction if the observation order of that variable is randomly permuted). Out of 50+ variables, one can focus on top 20 such, to speed up the training and not drop the accuracy significantly.

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.6.3
```

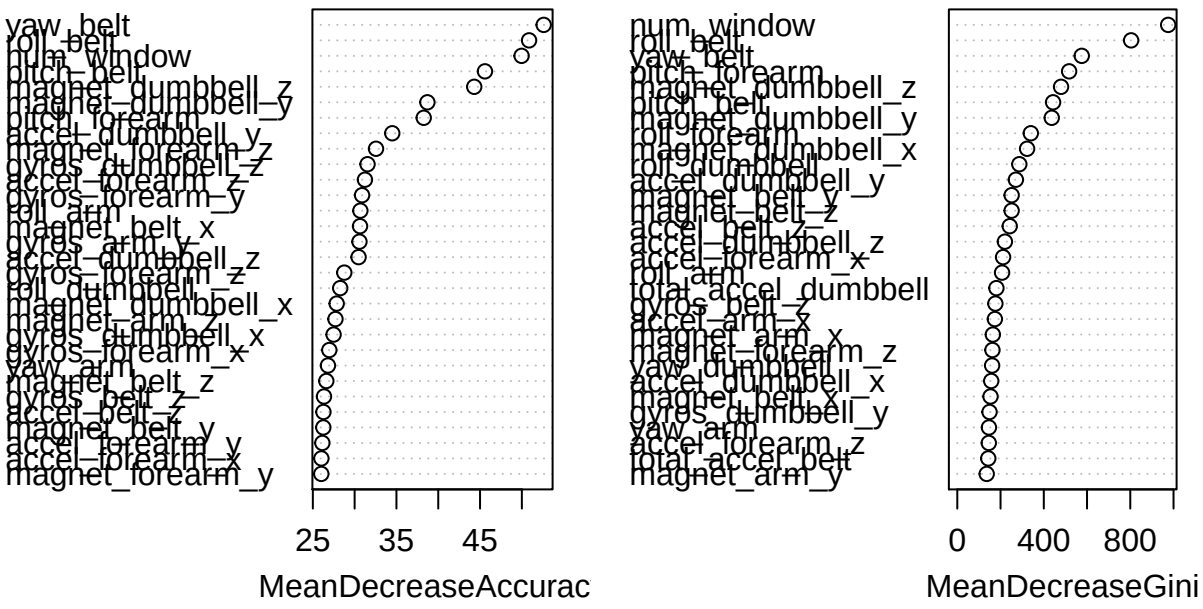
```
## randomForest 4.6-14
```

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

```
##
## Attaching package: 'randomForest'
```

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

```
frf <- randomForest(classe ~., data=training, importance = TRUE)
varImpPlot(frf)
```



One tries to combine these models to see if the result is better than each.

```
library(nnet)
```

```
## Warning: package 'nnet' was built under R version 3.6.3
```

```
predDF <- data.frame(prf=prf, pgbm=pgbm, classe=testing$classe)
combModFit <- multinom(classe ~., data = predDF)
```

```
## # weights:  50 (36 variable)
## initial  value 9471.542115
## iter  10 value 1781.110783
## iter  20 value 871.558668
## iter  30 value 461.308163
## iter  40 value 185.337668
## iter  50 value 73.774337
## iter  60 value 65.101324
## final  value 65.100255
## converged
```

```
combPred <- predict(combModFit, newdata = testing)
table(combPred, testing$classe)
```

```
##
```

```
## combPred    A    B    C    D    E
##        A 1674    2    0    0    0
##        B    0 1134    1    0    0
##        C    0    3 1025    1    0
##        D    0    0    0  963    3
##        E    0    0    0    0 1079
```

The result is the same as the random forest result. Therefore, we decide to use the random forest result as the final model fit for the validation set.

Finally, using the validation set of 20 observations, the prediction is as follows:

```
result <- predict(fitrf, newdata = validating)
result
```

```
##  [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
```