Analyzing exercise data

Diana Benavides

Sunday, May 24th, 2015

Overview

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 geek. 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. There is some popular dataset related to accelerometers on the belt, forearm, arm, and dumbell of 6 participants (available at *http://groupware.les.inf.puc-rio.br/har*). They were asked to perform barbell lifts correctly and incorrectly in 5 different ways.

This analysis will construct a model for describing and predicting body performance according to movements on the belt, forearm, arm and dumbell of any person, given a set of data.

Exploratory data analysis

We will use ***two different datasets***, one for training and other for testing our model. The first dataset is composed of 19622 rows and 160 columns; the final column contains our outcome variable, with the following body performance values (see Appendix, code chunk #1):

- 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)
- Throwing the hips to the front (Class E)

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

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

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

## Loading required package: lattice

## Warning: package 'randomForest' was built under R version 3.1.3
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

Class distribution for our training data is as shown below (see Appendix, code chunk #2):

Our testing dataset is composed of 20 rows and 160 columns, without any valid value for our outcome variable.

Data preparation

Since we have 160 variables involved in our training data, and we also have lots of null values in or test data, we need to perform several pre-processing steps in order to reduce the variable set as well as guarantying that our model will be able to predict test cases in a correct way. After applying operations such as elimination of variables in the training set for which there existed no value in the test set, replacing null values in the training set with mean values for the respective column, and standardizing our data, we finally obtain the following ***52 variables along with our class variable*** (see Appendix, code chunk #3):

```
##  [1] "roll_belt"            "pitch_belt"            "yaw_belt"
##  [4] "total_accel_belt"     "gyros_belt_x"          "gyros_belt_y"
##  [7] "gyros_belt_z"         "accel_belt_x"          "accel_belt_y"
## [10] "accel_belt_z"         "magnet_belt_x"         "magnet_belt_y"
## [13] "magnet_belt_z"        "roll_arm"              "pitch_arm"
## [16] "yaw_arm"              "total_accel_arm"       "gyros_arm_x"
## [19] "gyros_arm_y"          "gyros_arm_z"           "accel_arm_x"
## [22] "accel_arm_y"          "accel_arm_z"           "magnet_arm_x"
## [25] "magnet_arm_y"         "magnet_arm_z"          "roll_dumbbell"
## [28] "pitch_dumbbell"       "yaw_dumbbell"
"total_accel_dumbbell"
## [31] "gyros_dumbbell_x"     "gyros_dumbbell_y"      "gyros_dumbbell_z"
## [34] "accel_dumbbell_x"     "accel_dumbbell_y"      "accel_dumbbell_z"
## [37] "magnet_dumbbell_x"    "magnet_dumbbell_y"     "magnet_dumbbell_z"
## [40] "roll_forearm"         "pitch_forearm"         "yaw_forearm"
## [43] "total_accel_forearm"  "gyros_forearm_x"       "gyros_forearm_y"
## [46] "gyros_forearm_z"      "accel_forearm_x"       "accel_forearm_y"
## [49] "accel_forearm_z"      "magnet_forearm_x"      "magnet_forearm_y"
## [52] "magnet_forearm_z"     "classe"
```

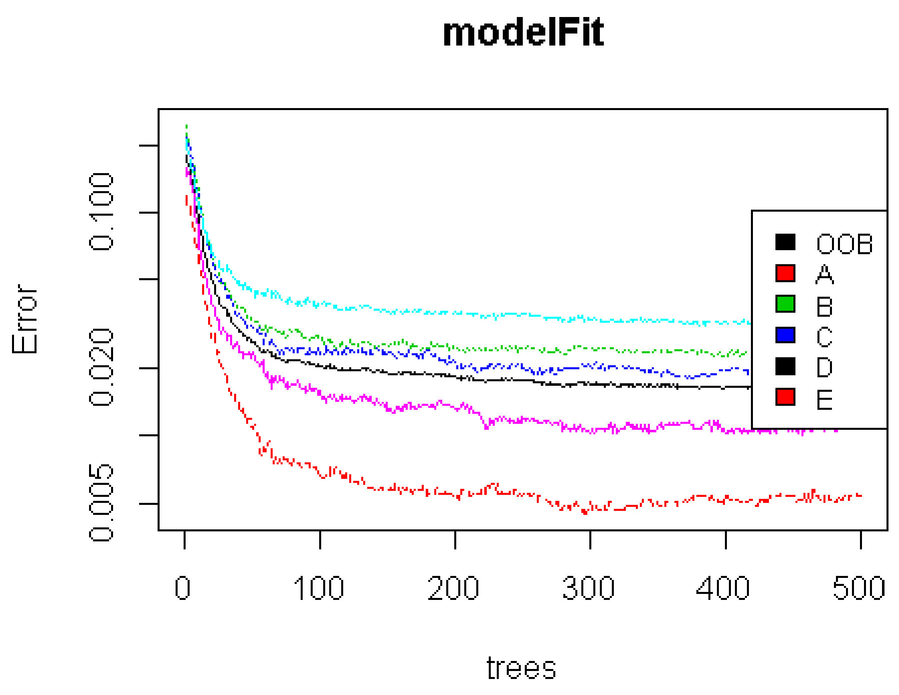
As a second pre-processing step, we performed ***principal-component analysis (PCA)***, in order to obtain a reduced set of variables for our model (see Appendix, code chunk #4).

Prediction model

For our prediction model, and because of its theoretical high accuracy, we select ***random forests*** as our technique. We apply repeated cross-validation with the idea of obtaining a more accurate and less overfitted model. Our model accuracy resulted to be very high (see Appendix, code chunk #5):

```
##
## Call:
##  randomForest(formula = finalTraining$classe ~ ., data = trainPC,
trControl = fitControl)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 1.65%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 5550    8   14    5    3 0.005376344
## B   52 3709   30    0    6 0.023176192
## C    5   32 3361   23    1 0.017825833
## D    6    0   93 3113    4 0.032027363
## E    0    8   20   14 3565 0.011644026
```

A graphic representation of our random forests model is as shown below (see Appendix, code chunk #6):



We test it with our testing data, for which we don't know classes values (we assigned class A only for testing purposes), and we obtain the following predictions (see Appendix, code chunk #7):*

```
##
##     A B C D E
##   A 8 5 2 3 2
```

*We obtained 19/20 correct predictions in the programming assignment presented along with this report, thus obtaining an out-of-sample error of only 0.05.

Conclusion and final remarks

As part of this experiment, ***we built a random forests model for predicting body performance according to a set of measures over human body***. Our model implied, besides availability of data, several pre-processing steps such as removing nulls, standardizing variables and performing PCA.

APPENDIX

- Code chunk #1

```
library(ISLR)
library(ggplot2)
library(caret)
library(randomForest)

#READING DATA
training<-read.csv("pml-training.csv")
validation<-read.csv("pml-testing.csv")
```

- Code chunk #2

```
counts <- table(training$classe)
barplot(counts,main="Weight Lifting Exercise", names.arg=c("A", "B", "C",
"D", "E"), col=c("green", "blue", "yellow", "orange", "red"))
```

- Code chunk #3

```
#PRE-PROCESSING DATA

#Divide by numeric and factor variables
numericValidation<-validation[sapply(validation,is.numeric)]
vars<-colnames(training) %in% colnames(numericValidation)

numericTraining <- training[vars]

vars2<-colnames(numericValidation) %in% colnames(numericTraining)
numericValidation <- numericValidation[vars2]
```

```r
factorTraining <- training[sapply(training,is.factor)]

factorValidation <- validation[sapply(validation,is.factor)]

#Replace missing values
#columns and their means
for(i in 1:ncol(numericTraining)){
  numericTraining[is.na(numericTraining[,i]), i] <-
mean(numericTraining[,i], na.rm = TRUE)
  numericValidation[is.na(numericTraining[,i]), i] <-
mean(numericTraining[,i], na.rm = TRUE)
}

#Since most of the variables have large standard deviations,
standardizing all numeric variables
preObj<-preProcess(numericTraining,method=c("center","scale"))
numericTraining<-predict(preObj, numericTraining, )

preObj<-preProcess(numericValidation,method=c("center","scale"))
numericValidation<-predict(preObj, numericValidation, )

#Since all of the factor variables seem to be noise, or identifiers, they
are removed from analysis
finalTraining<-cbind.data.frame(numericTraining,factorTraining$classe)
colnames(finalTraining)[57]<-"classe"
finalTraining<-as.data.frame(finalTraining)

#Remove useless columns
finalTraining<-finalTraining[,-1]
finalTraining<-finalTraining[,-1]
finalTraining<-finalTraining[,-1]
finalTraining<-finalTraining[,-1]

finalValidation<-as.data.frame(numericValidation)
#finalValidation<-as.numeric(finalValidation)
finalValidation$classe<-
c("A","A","A","A","A","A","A","A","A","A","A","A","A","A","A","A","A","A"
,"A","A")
finalValidation$classe<-as.factor(finalValidation$classe)
finalValidation<-finalValidation[,-1]
finalValidation<-finalValidation[,-1]
finalValidation<-finalValidation[,-1]
finalValidation<-finalValidation[,-1]

names(finalValidation)<-names(finalTraining)

#Our final variables are...
names(finalTraining)
```

- Code chunk #4

```r
#Preprocessing
preProc<-preProcess(finalTraining[,-53], method="pca", na.remove=T)
trainPC<-predict(preProc, finalTraining[,-53])
```

- Code chunk #5

```r
#CV
fitControl <- trainControl(method = "repeatedcv",number = 10,repeats =
10)

#PREDICTING (with PCA)
set.seed(825)
modelFit<-randomForest(finalTraining$classe ~ ., data=trainPC,
trControl=fitControl)
modelFit
```

- Code chunk #6

```r
plot(modelFit, log='y')
legend("right", colnames(modelFit$err.rate),col=1:4,cex=0.8,fill=1:4)
```

- Code chunk #7

```r
testPC<-predict(preProc, finalValidation[,-53])
table(finalValidation$classe, predict(modelFit, testPC))
```