Prediction Assignment: Classifying Activity Class

Sophie Yang

4/16/2017

Introduction

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, the goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants to predict the manner in which they did the exercise. This is the "classe" variable in the training set. This report describes how the model was built.

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, the participants 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 (http://groupware.les.inf.puc-rio.br/har) (see the section on the Weight Lifting Exercise Dataset).

Datasets: Training and Testing

```
library(dplyr); library(ggplot2); library(lattice);
library(caret); library(gbm); library(survival);
library(randomForest); library(MASS); library(mice);
library(plyr);library(reshape2);library(rattle);
```

```r
# Download training and testing data
fileURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
download.file(fileURL, destfile = "training.csv", method = "curl")
training <- read.csv("training.csv")

fileURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
download.file(fileURL, destfile = "testing.csv", method = "curl")
testing <- read.csv("testing.csv")

set.seed(12345)

# Cleaning the Data:
# 1. Use only the variables in the dataset without NAs and remove unnecessary columns
train <- training[which(colSums(is.na(training)) == 0)]
train <- train[,c(8: dim(train)[2])]

# 2. We still have too many variables at hand, so we use PCA to narrow it down
preProcPCA <- preProcess(train, method = "pca", thresh = 0.99)
trainPCA <- predict(preProcPCA, train)
trainPCA <- data.frame(classe = trainPCA$classe, trainPCA[grep("^PC",names(trainPCA))])
head(cbind(trainPCA[,1], round(trainPCA[,2:dim(trainPCA)[2]],4)))
```

```
##   trainPCA[, 1]    PC1    PC2     PC3    PC4     PC5    PC6     PC7
## 1             A 4.3383 1.6670 -2.7796 0.8328 -1.2983 2.0634 -0.1544
## 2             A 4.3806 1.6748 -2.7808 0.8367 -1.3705 2.1368 -0.1991
## 3             A 4.3494 1.6933 -2.7809 0.8336 -1.3039 2.0718 -0.1822
## 4             A 4.3630 1.6854 -2.7669 0.8320 -1.3163 2.1099 -0.2093
## 5             A 4.3517 1.7438 -2.7374 0.8228 -1.3286 2.1529 -0.2241
## 6             A 4.3590 1.7151 -2.7787 0.8299 -1.3132 2.0926 -0.1914
##       PC8     PC9    PC10   PC11    PC12    PC13   PC14    PC15   PC16
## 1 -2.7435 -0.0299 -0.2627 0.7142 -0.9333 -2.9430 0.2493 -0.1556 0.8967
## 2 -2.6911  0.0041 -0.2948 0.6979 -0.8412 -2.9300 0.2910 -0.0571 0.8727
## 3 -2.7190  0.0032 -0.2844 0.7049 -0.8602 -2.9241 0.2749 -0.1159 0.8888
## 4 -2.6871  0.0253 -0.2931 0.7133 -0.8876 -2.9215 0.3143 -0.0717 0.8800
## 5 -2.6939 -0.0197 -0.3159 0.6569 -0.8485 -2.9097 0.1845 -0.1082 0.8193
## 6 -2.7238  0.0058 -0.2664 0.6923 -0.8971 -2.9285 0.2894 -0.1082 0.8805
##      PC17    PC18   PC19    PC20    PC21   PC22   PC23    PC24   PC25
## 1 -0.7024 -0.3969 1.3858 -0.4614 -0.4536 0.2462 0.2963  0.0364 0.3622
## 2 -0.7208 -0.3623 1.3935 -0.4589 -0.4059 0.1984 0.2936 -0.0184 0.3657
## 3 -0.7145 -0.3842 1.3846 -0.4405 -0.4392 0.2301 0.2794  0.0446 0.3757
## 4 -0.6830 -0.3661 1.3853 -0.4523 -0.3998 0.2266 0.3348  0.0324 0.3816
## 5 -0.7410 -0.3685 1.3716 -0.4696 -0.4165 0.2444 0.2952  0.0308 0.3750
## 6 -0.6945 -0.3841 1.3719 -0.4512 -0.4199 0.2497 0.2855  0.0364 0.3989
##      PC26    PC27   PC28    PC29    PC30   PC31    PC32    PC33   PC34
## 1 -0.0532 -1.0009 0.1034 -0.0646 -0.0269 0.1369 -0.6165 -0.2944 0.1957
## 2 -0.0606 -0.9987 0.1015 -0.1189 -0.1669 0.1301 -0.5689 -0.2920 0.1425
## 3 -0.0387 -1.0166 0.1360 -0.0586  0.0234 0.1335 -0.6235 -0.2824 0.2040
## 4 -0.0496 -0.9930 0.0954 -0.0919 -0.0892 0.1318 -0.5812 -0.3019 0.1250
## 5 -0.0350 -1.0233 0.0982 -0.1081  0.0779 0.1235 -0.6236 -0.4167 0.0760
## 6 -0.0252 -0.9825 0.1081 -0.0886 -0.0872 0.1082 -0.5817 -0.3134 0.1667
##      PC35    PC36
## 1 -0.0163 -0.1928
## 2 -0.0366 -0.2022
## 3 -0.0212 -0.2233
## 4 -0.0459 -0.2341
## 5 -0.0035 -0.2724
## 6  0.0012 -0.1957
```

```
# 3. Next we train our model using LDA, since we have multiple classes
modelRF <- randomForest(classe~., data = trainPCA)

# We can visualize 1 of the random trees from the random trees we generated
tree <-getTree(modelRF, k = 1, labelVar = TRUE)
head(tree)
```

```
##   left daughter right daughter split var split point status prediction
## 1             2              3      PC14 -0.89924155      1       <NA>
## 2             4              5      PC15  0.10313183      1       <NA>
## 3             6              7      PC30  0.48264321      1       <NA>
## 4             8              9      PC16 -2.58657347      1       <NA>
## 5            10             11      PC35  0.05895105      1       <NA>
## 6            12             13       PC8  1.35730583      1       <NA>
```

```
tail(tree)
```

```
##      left daughter right daughter split var split point status prediction
## 3268          3272           3273       PC9  -0.3833826      1       <NA>
## 3269             0              0      <NA>   0.0000000     -1          C
## 3270             0              0      <NA>   0.0000000     -1          D
## 3271             0              0      <NA>   0.0000000     -1          C
## 3272             0              0      <NA>   0.0000000     -1          C
## 3273             0              0      <NA>   0.0000000     -1          D
```

As we can see, the top is the start of the tree, whose outputs are NAs, which makes sense because they lead to downstream branches. The tails as an example are the lowest part of the notes, which, as expected show the eventual classes they are classified to.

```
# 4. CLean up testing data
test <- testing[which(colSums(is.na(training)) == 0)]
testPCA <- predict(preProcPCA, test)
testPCA <- data.frame(testPCA[grep("^PC",names(testPCA))])
```

Analysis

In our analysis above we accomplished a creating a model to test our testing data. Below we test the result of the model against our test data.

```
testPred <- cbind(classe = predict(modelRF, testPCA), test)
testPred[,1:3]
```

```
##    classe  X user_name
## 1       B  1     pedro
## 2       A  2    jeremy
## 3       B  3    jeremy
## 4       A  4    adelmo
## 5       A  5    eurico
## 6       E  6    jeremy
## 7       D  7    jeremy
## 8       B  8    jeremy
## 9       A  9  carlitos
## 10      A 10   charles
## 11      B 11  carlitos
## 12      C 12    jeremy
## 13      B 13    eurico
## 14      A 14    jeremy
## 15      E 15    jeremy
## 16      E 16    eurico
## 17      A 17     pedro
## 18      B 18  carlitos
## 19      B 19     pedro
## 20      B 20    eurico
```