Machine Learning Project - Sensor Data

Shubhadeep Purkayastha

3/16/2017

Introduction

Using devices such as Jawbone Up, Nike Fuelband, and Fitbit it is now possible to collect a large amount of data about personal activity. 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 or to find patterns in their behavior.

However, rarely do people quantify how well they do a particular activity. In this project, we will use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants, who were asked to perform barbell lifts correctly and incorrectly in 5 different ways.

Data for this project comes from http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har) (http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har))

Six participants performed 10 bicep curls in five different fashions: exactly according to the correct 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).

The goal of this project is to predict how well a bicep curl was performed using the data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants.

Data Processing

The data appears at first to have a lot of NA values. The data is provided with aggregated statistical metrics across each window of observation. The columns that contain these aggregated values are assigned NA while the data is collected. For this analysis I chose to separate the aggregated data and the raw data into 2 data frames and build models off of both. The section of code below includes creating a training/testing partition and separating the aggregated data columns from the raw data and finally the removal of the NA values from the summarized data.

Loading the required libraries

```
suppressPackageStartupMessages(library(AppliedPredictiveModeling))
suppressPackageStartupMessages(library(caret))
suppressPackageStartupMessages(library(kernlab))
suppressPackageStartupMessages(library(randomForest))
```

Download full data for partitioning

```
fileUrl <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
download.file(fileUrl, destfile= "./Sensordata/training.csv", method = "curl")
data <- read.csv("./Sensordata/training.csv", sep=",", header=TRUE,
                 na.strings = c("#DIV/0!", "NA", "N/A", "null", "?"))
head(data)
names(data)
dim(data)
```

Download 20 Case data for validation

```
fileUrl <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
download.file(fileUrl, destfile= "./sensordata/testing.csv", method = "curl")
validation <- read.csv("./Sensordata/testing.csv", sep=",", header=TRUE,
                       na.strings = c("#DIV/0!", "NA", "N/A", "null", "?"))
head(validation)
dim(validation)
```

Exploring the dataset and the predictor variable to be used

```
summary(data)
```

```
dim(data)
```

```
## [1] 19622   160
```

```
class(data$classe)
```

```
## [1] "factor"
```

```
table(data$classe)
```

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

```
table(data$new_window)
```

```
##
##    no   yes
## 19216   406
```

Determining the quantity of NA values in the data

```
NA.levels <- unique(apply(data, 2,function(x){sum(is.na(x))}))
number.NA <- dim(data)[1]- NA.levels[2]
non.NA <- number.NA/dim(data)[1]
sprintf("%1.2f%%", 100*non.NA)
```

```
## [1] "2.02%"
```

Setting empty spaces and div0 values to NA

```
data[data == ""] <- NA
data[data == "#DIV/0!"] <- NA
data[data == "<NA>"] <- NA
```

Partitioning the data 80/20 and creating the training & testing datasets

```
suppressPackageStartupMessages(library(AppliedPredictiveModeling))
suppressPackageStartupMessages(library(caret))
set.seed(22)
inTrain <- createDataPartition(data$classe, p=0.8, list=FALSE)
training <- data[inTrain,]
testing <- data[-inTrain,]
names(training)
dim(training)
dim(testing)
```

```
table(training$new_window)
```

```
##
##    no   yes
## 15361   338
```

Selecting non-aggregated sensor data

```
train_raw <- training[which(training$new_window == "no"),]
test_raw <- testing[which(testing$new_window == "no"),]
train_raw <- train_raw[!colSums(is.na(train_raw)) > 0]
table(train_raw$new_window)
```

```
##
##    no   yes
## 15361     0
```

```
sum(is.na(train_raw))
```

```
## [1] 0
```

Selecting aggregated sensor data

```
train_agg <- training[which(training$new_window == "yes"),]
test_agg <- testing[which(testing$new_window == "yes"),]
table(train_agg$new_window)
```

```
##
##  no yes
##   0 338
```

```
table(test_agg$new_window)
```

```
##
##  no yes
##   0  68
```

Some more pre-processing of the aggregated data before model fitting

Removing data columns contaning NA or zero values

```
train_agg.clean <- subset(train_agg,
                      select=-c(kurtosis_yaw_belt,skewness_yaw_belt,amplitude_yaw_belt
,
                      kurtosis_yaw_dumbbell,skewness_yaw_dumbbell,amplitude_yaw_dumbbe
ll,
                      kurtosis_yaw_forearm,skewness_yaw_forearm,amplitude_yaw_forearm)
)

test_agg.clean <- subset(test_agg,
                      select=-c(kurtosis_yaw_belt,skewness_yaw_belt,amplitude_yaw_belt
,
                      kurtosis_yaw_dumbbell,skewness_yaw_dumbbell,amplitude_yaw_dumbbe
ll,
                      kurtosis_yaw_forearm,skewness_yaw_forearm,amplitude_yaw_forearm)
)
```

Removing rows containing NA values & confirming the purity of the data

```
train.final <- train_agg.clean[complete.cases(train_agg.clean),]
sum(is.na(train.final))
```

```
## [1] 0
```

```
test.final <- test_agg.clean[complete.cases(test_agg.clean),]
sum(is.na(test.final))
```

```
## [1] 0
```

Model Fitting

A random forest (RF) model is best suited for model fitting in this scenario because the sensor data has a lot of noise. The RF model uses bootstrap resampling with the training set partition given above to crossvalidate against the testing set. Since the fit uses all the possible (59) clean predictor variables, k-fold cross validation would be computationally intensive and not suitable towards the present purpose.

Model Fitting - Model 1

Not including the index X row in the dataset. The data is organized alphabetically by class outcome.

```
library(randomForest)
model1 <- randomForest(classe~. , data=train_raw[,-c(1:7)], method="class")
model1
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = train_raw[, -c(1:7)],      method = "cl
ass")
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.46%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 4371    2    0    0    1 0.0006858711
## B   14 2952    7    0    0 0.0070635721
## C    0   12 2665    2    0 0.0052258305
## D    0    0   25 2486    1 0.0103503185
## E    0    1    0    5 2817 0.0021253985
```

```
pred_test1 <- predict(model1, testing)
pred_train1 <- predict(model1, training)

confusionMatrix(pred_test1, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1115    3    0    0    0
##          B    0  756    3    0    0
##          C    0    0  681    3    0
##          D    0    0    0  639    3
##          E    1    0    0    1  718
##
## Overall Statistics
##
##                Accuracy : 0.9964
##                  95% CI : (0.994, 0.998)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9955
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9991   0.9960   0.9956   0.9938   0.9958
## Specificity            0.9989   0.9991   0.9991   0.9991   0.9994
## Pos Pred Value         0.9973   0.9960   0.9956   0.9953   0.9972
## Neg Pred Value         0.9996   0.9991   0.9991   0.9988   0.9991
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2842   0.1927   0.1736   0.1629   0.1830
## Detection Prevalence   0.2850   0.1935   0.1744   0.1637   0.1835
## Balanced Accuracy      0.9990   0.9975   0.9973   0.9964   0.9976
```

```
confusionMatrix(pred_train1, 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    1    0
##          D    0    0    0 2572    0
##          E    0    0    0    0 2886
##
## Overall Statistics
##
##                Accuracy : 0.9999
##                  95% CI : (0.9996, 1)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9999
##  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   0.9996   1.0000
## Specificity            1.0000   1.0000   0.9999   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   0.9996   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   0.9999   1.0000
## Prevalence             0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1935   0.1744   0.1638   0.1838
## Detection Prevalence   0.2843   0.1935   0.1745   0.1638   0.1838
## Balanced Accuracy      1.0000   1.0000   1.0000   0.9998   1.0000
```

Model Fitting - Model 2

The second model (model2) uses feature selection to narrow down the 59 predictors to only 7 chosen variables: classe ~ roll_belt + pitch_belt + yaw_belt + magnet_arm_ + gyros_dumbbell_y + magnet_dumbbell_y + pitch_forearm. To create this model, 3-fold crossvalidation was implemented with the caret package.

Using Correlation based feature selection and best-first algorithm

```
suppressPackageStartupMessages(library(caret))
suppressPackageStartupMessages(library(FSelector))

feature.select <- cfs(classe ~.,train_raw.clean[,-c(1:7)])
f <- as.simple.formula(feature.select, "classe")

fitControl <- trainControl(method = "cv", number = 3, repeats = 3)
model2 <- train(f, method = "rf", data =train_raw.clean, trControl = fitControl)
model2
```

```
##
## Call:
## Random Forest
##
## 15699 samples
##     7 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 10466, 10465, 10467
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.9761127  0.9697912
##   4     0.9738197  0.9668918
##   7     0.9661761  0.9572249
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 22
```

```
pred_test2 <- predict(model2, testing)
pred_train2 <- predict(model2, training)

confusionMatrix(pred_test2, testing$classe)
```

```
confusionMatrix(pred_train2, training$classe)
```

Model Fitting - Model 3

The final model (model3) is fit using the provided summary data.

To create this model, 3-fold crossvalidation was implemented with the caret package.

Using summary statistics for prediction

```
suppressPackageStartupMessages(library(caret))
suppressPackageStartupMessages(library(FSelector))

features3 <- cfs(classe ~.,train.final[,-c(1:7)])
z <- as.simple.formula(features3, "classe")

model3 <- train(z, method = "rf", data =train.final, trControl = fitControl)
model3
```

```
##
## Call:
## Random Forest
##
## 187 samples
##  11 predictor
##   5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 124, 126, 124
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##      2  0.2567439  0.0000000
##     53  0.6462833  0.5482757
##   1456  0.6782028  0.5940114
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 1456.
```

```
pred_test3 <- predict(model3, test.final)
pred_train3 <- predict(model3, train.final)

confusionMatrix(pred_test3, test.final$classe)
```

```
confusionMatrix(pred_test3, train.final$classe)
```

Conclusions

The first model (model1) achieves an estimated out of sample error rate of 0.46%.

The second model (model2) achieves a 97.61% accuracy with an expected error of 2.31%. The expected error is higher, but is still very successful considering this model uses 52 fewer predictors (used only 7 selected predictors).

The third model achieves an accuracy of only 71.74%, or a 28.26% expected error rate against the test validation set. Therefore, using the summary statistics for

prediction leads to reduced accuracy. The most differetiating variable is the belt sensor, which quickly distinguishes a number of cases where the individual commits a class E mistake ("throwing the hips to the front").

The first model performed the best overall and will be used to predict the validation test set.

```
predict(model1,validation)
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```

APPENDIX

Ugulino, W.; Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H. Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements. Proceedings of 21st Brazilian Symposium on Artificial Intelligence. Advances in Artificial Intelligence - SBIA 2012. In: Lecture Notes in Computer Science. , pp. 52- 61. Curitiba, PR: Springer Berlin / Heidelberg, 2012. ISBN 978-3-642-34458-9. DOI: 10.1007/978-3-642- 34459-6_6.

L. B. Statistics and L. Breiman. Random forests. In Machine Learning, pages 5–32, 2001.

Read more: http://groupware.les.inf.puc-rio.br/har#sbia_paper_section#ixzz4NpZpLz5s (http://groupware.les.inf.puc-rio.br/har#sbia_paper_section#ixzz4NpZpLz5s) (http://groupware.les.inf.puc-rio.br/har#sbia_paper_section#ixzz4NpZpLz5s (http://groupware.les.inf.puc-rio.br/har#sbia_paper_section#ixzz4NpZpLz5s))