Predicting wellness of physical exercise (weight lifting)

Andra

28 Feb 2016

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, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They 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 (see the section on the Weight Lifting Exercise Dataset).

Scope

The main purpose of this project is to model how well individuals are exercising and to predict using the model how well new individuals will train using weights. The outcome (classe in our dataset) that we are trying to predict can take 5 values : A, B, C, D and E.

Data Used

The data for this project come from this source: http://groupware.les.inf.puc-rio.br/har.

The training data for this project are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

The test data are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

```
trainUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

#reading training and testing data and marking NA or empty fields as NA
training <- read.csv(url(trainUrl), na.strings=c("NA","","  "))
testing <- read.csv(url(testUrl), na.strings=c("NA","  ",""))

dim(training)
```

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

```
dim(testing)
```

```
## [1]  20 160
```

Data Cleaning

```
noofnas<-apply(training, 2, function(x) length(which(is.na(x)))) # calculating the number of NA variabl
threshold<-0.7*nrow(training) # set threshold for variables to be removed: variables that have more tha

noofnas<-subset(noofnas,noofnas<=threshold) # removing variables that do not fulfill condition above fr

training<-training[,names(noofnas)] # removing the variables that do not fulfill the % of NA condition
dim(training)
```

```
## [1] 19622    60
```

```
timestamps.columns<-names(training)[3:5] # removing timestamps from model; timestamp is not a valid var

training<-training[, -which(names(training) %in% c("X","user_name", timestamps.columns))]
dim(training)
```

```
## [1] 19622    55
```

```
temp<- training[, -which(names(training) %in% c("X","user_name","classe"))] # creating temp data frame
testing<-testing[,c(names(temp),"problem_id")]

dim(testing)
```

```
## [1] 20 55
```

Next, we need to make sure that the type (class) for each of the variables in the two data frames (training and testing) are the same . We consider data until length-1, as the last element is the prediction class and problem_id respectively . In order to do that, we are sorting the columns from the two datasets and then compare the types for each of the columns. For the ones that are different, we assign the right type. This will ensure that we will not have any unpleasant suprises when running the algorithms.

```
temp<-training[, 1:ncol(training)-1]
temp<-temp[, order(names(temp))]
training<-cbind(temp, training$classe)
colnames(training)[length(training)]<-"classe"


temp<-testing[, 1:ncol(testing)-1]
temp<-temp[, order(names(temp))]
testing<-cbind(temp, testing$problem_id)
colnames(testing)[length(testing)]<-"problem_id"


classes.testing<-lapply(testing,class)
classes.training<-lapply(training,class)

classes.testing<-unlist(classes.testing)
classes.training<-unlist(classes.training)
```

```
test.class<-classes.testing[1:length(classes.testing)-1]==classes.training[1:length(classes.training)-1]
dif<-names(test.class[test.class==FALSE])

print(dif)
```

```
## [1] "magnet_dumbbell_z" "magnet_forearm_y"  "magnet_forearm_z"
```

```
## there are 3 columns with a different type in the training set compared to the testing set
##separate test has showed that the columns in the testing set are "integer", whereas in the training se
#we thus assign the right type

for (i in 1: length(dif))
   class(training[,dif[i]])<-"integer"
```

Partitioning training data set

Further, we are partitioning our training data set into training set (60%) for fitting the model and testing set (40%) for testing our model .

```
library(caret)
library(rattle)
library(rpart)
library(rpart.plot)
library (MASS)
library(pgmm)
library(randomForest)
library(e1071)
library(kernlab)
```

```
inTrain <- createDataPartition(y=training$classe, p=0.6, list=FALSE)
training.new <- training[inTrain, ]
testing.new <- training[-inTrain, ]
dim(training.new)
```

```
## [1] 11776    55
```

```
dim(testing.new)
```

```
## [1] 7846   55
```

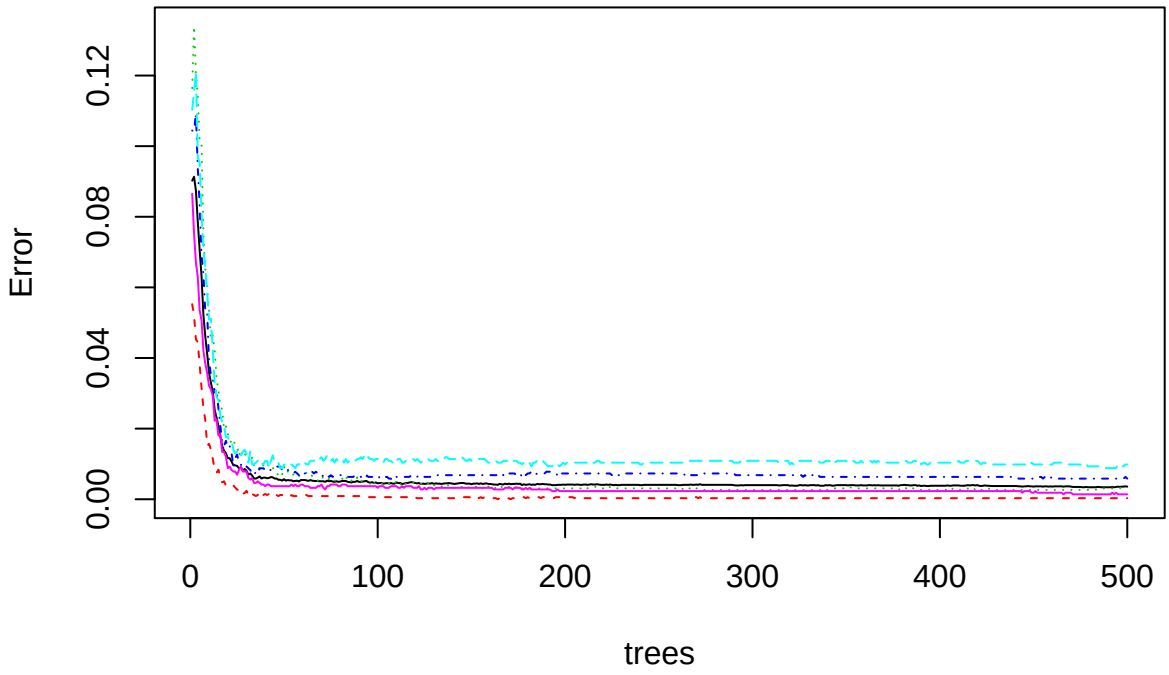
```
levels(testing$new_window ) <- levels(training.new$new_window) #ensure the same levels for the factor v
```

Prediction with Random Forrest

```r
set.seed(12345)

# Fitting the model
rf <- randomForest(classe ~ ., data=training.new, method="class")


# testing the model on our prediction data set

mypredictions<- predict(rf,  testing.new, type = "class")

# Results from testing the model : predicted versus observed
cfx<-confusionMatrix(mypredictions, testing.new$classe)
cfx
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    2    0    0    0
##          B    0 1515    5    0    0
##          C    0    1 1362    8    0
##          D    0    0    1 1277    0
##          E    0    0    0    1 1442
##
## Overall Statistics
##
##                Accuracy : 0.9977
##                  95% CI : (0.9964, 0.9986)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9971
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9980   0.9956   0.9930   1.0000
## Specificity            0.9996   0.9992   0.9986   0.9998   0.9998
## Pos Pred Value         0.9991   0.9967   0.9934   0.9992   0.9993
## Neg Pred Value         1.0000   0.9995   0.9991   0.9986   1.0000
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1931   0.1736   0.1628   0.1838
## Detection Prevalence   0.2847   0.1937   0.1747   0.1629   0.1839
## Balanced Accuracy      0.9998   0.9986   0.9971   0.9964   0.9999
```

```r
plot(rf, main="Random Forest Fit")
```



Results :

```
cfx$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9977058      0.9970980      0.9963766      0.9986398      0.2844762
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

Prediction with Support Vector Machines

```
set.seed(12345)
# Fitting the model
sv <- svm(classe ~ ., data=training.new, method="class")


# testing the model on our prediction data set

mypredictions<- predict(sv,  testing.new, type = "class")

# Results from testing the model : predicted versus observed
cfx<-confusionMatrix(mypredictions, testing.new$classe)
cfx
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction    A    B    C    D    E
##          A 2205   98    4    5    0
##          B    5 1372   53    0    5
##          C   21   46 1283  120   39
##          D    0    1   27 1161   31
##          E    1    1    1    0 1367
##
## Overall Statistics
##
##                Accuracy : 0.9416
##                  95% CI : (0.9362, 0.9467)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9261
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9879   0.9038   0.9379   0.9028   0.9480
## Specificity            0.9809   0.9900   0.9651   0.9910   0.9995
## Pos Pred Value         0.9537   0.9561   0.8502   0.9516   0.9978
## Neg Pred Value         0.9951   0.9772   0.9866   0.9811   0.9884
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2810   0.1749   0.1635   0.1480   0.1742
## Detection Prevalence   0.2947   0.1829   0.1923   0.1555   0.1746
## Balanced Accuracy      0.9844   0.9469   0.9515   0.9469   0.9738
```

Results :

```
cfx$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   9.416263e-01   9.260789e-01   9.362087e-01   9.467129e-01   2.844762e-01
## AccuracyPValue  McnemarPValue
##   0.000000e+00   3.911321e-44
```

Prediction with Regression Trees

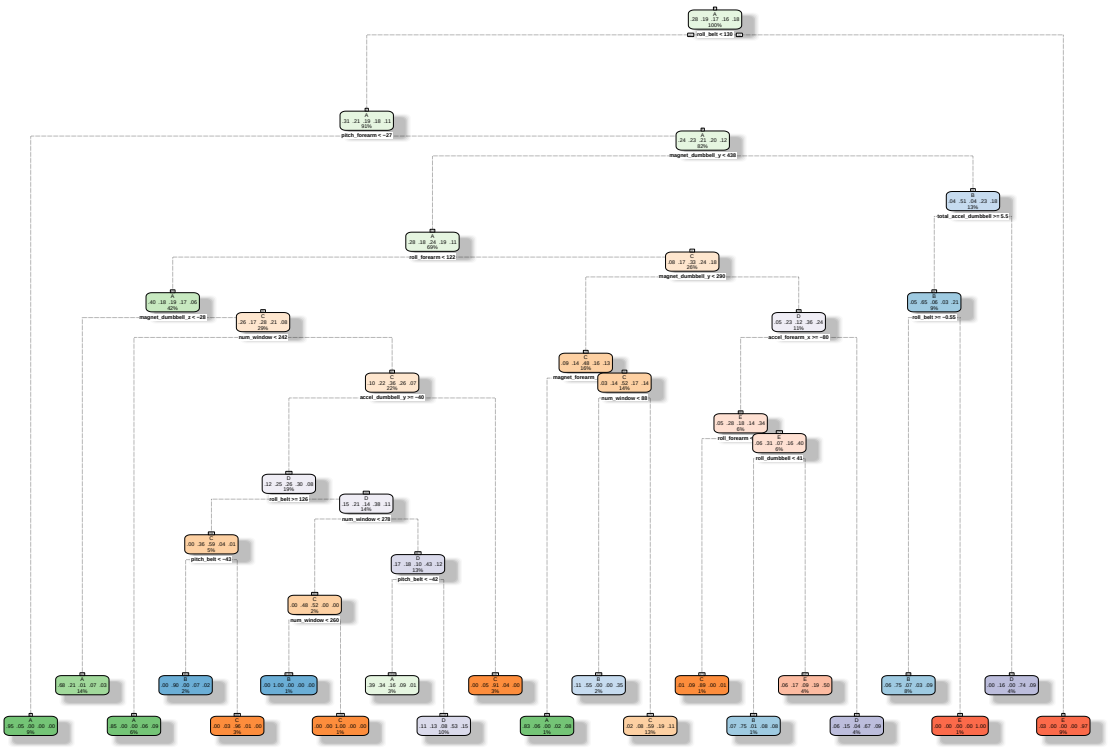
```
set.seed(12345)
# Fitting the model
rp <- rpart(classe ~ ., data=training.new, method="class")

# testing the model on our prediction data set

mypredictions<- predict(rp,  testing.new, type = "class")

# Results from testing the model : predicted versus observed
```

```
cfx<-confusionMatrix(mypredictions, testing.new$classe)
cfx
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1983  358   60  140   87
##          B   52  819   41   25  111
##          C   29  103 1151  185  112
##          D  114  183   86  877  162
##          E   54   55   30   59  970
##
## Overall Statistics
##
##                Accuracy : 0.7392
##                  95% CI : (0.7294, 0.7489)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6685
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8884   0.5395   0.8414   0.6820   0.6727
## Specificity            0.8851   0.9638   0.9338   0.9169   0.9691
## Pos Pred Value         0.7546   0.7815   0.7285   0.6167   0.8305
## Neg Pred Value         0.9523   0.8972   0.9654   0.9363   0.9293
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2527   0.1044   0.1467   0.1118   0.1236
## Detection Prevalence   0.3349   0.1336   0.2014   0.1812   0.1489
## Balanced Accuracy      0.8868   0.7517   0.8876   0.7994   0.8209
```

```
# View our tree
fancyRpartPlot(rp)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Rattle 2016−Feb−28 23:42:18 andratolbus

Results :

```
cfx$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   7.392302e-01   6.685252e-01   7.293637e-01   7.489186e-01   2.844762e-01
## AccuracyPValue  McnemarPValue
##   0.000000e+00  4.576929e-111
```

Conclusion

By looking at the accuracy values for the three models, we can conclude that for this data set Random Forest (99%) has performed the best, followed closely by SVM (Accuracy 95%). The accuracy for the regression tree model has an accuracy of only approximately 80%. The expect out of sample error for Random Trees is 100-99.77=0.23%

Test on the prediction data set

```
predict(rf,  testing, type = "class")
```

```
##  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
```