Machine Learning Final Project

Roberto Rojas Esteban

12-10-2020

Overview

With devices such as the Jawbone Up, Nike FuelBand, and Fitbit, it is possible to collect a large amount of personal activity data considerably inexpensively. These types of devices are part of quantified self-movement: a group of users who take action on themselves regularly to improve their health, to find patterns in their behavior, or because they are fans of technology and physical exercise. One thing that people do on a regular basis is quantify how much of a particular activity they do, but they rarely quantify how well they do it.

In this project, we will use accelerometer data on the belt, forearm, arm, and dumbbell from 6 participants. They were asked to perform correctly and incorrectly barbell lifts in 5 different ways.

The data consists of two types of data sets: training and test data (which will be used to validate the selected model).

The central idea of the project is to predict how they performed the exercise. This is the variable "class" in the training set. You can use any of the other variables to predict. Our goal is to predict the labels of the observations in the test set.

Below is the code that was used to create the model, estimate the out-of-sample error, and make predictions. A description of each step in the process is also included.

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).

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

The data for this project come from this source: http://web.archive.org/web/20161224072740/http://groupware.les.inf.puc-rio.br/har. If you use the document you create for this class for any purpose please cite them as they have been very generous in allowing their data to be used for this kind of assignment.

Load Packages and Data

```
library(caret)
```

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

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

```
library(ggplot2)
library(rattle)
```

```
## Loading required package: tibble
```

```
## Loading required package: bitops
```

```
## Rattle: A free graphical interface for data science with R.
## Versión 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Escriba 'rattle()' para agitar, sacudir y  rotar sus datos.
```

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
library(lattice)
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
```

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

```
library(randomForest)
```

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

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

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

```
## The following object is masked from 'package:rattle':
##
##     importance
```

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

```
library(rpart)
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
if (!file.exists("MachineLearning")){
  dir.create("MachineLearning")
  url_train<-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
  url_test<-"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
  download.file(url_train, destfile = "./MachineLearning/pml-training.csv")
  download.file(url_test, destfile="./MachineLearning/pml-testing.csv")
}
  traindata<-read.csv("./MachineLearning/pml-training.csv")
  testdata<-read.csv("./MachineLearning/pml-testing.csv")
```

We see that there are 160 variables and 19622 observations in the training set, while 20 observations for the test set.

Cleaning the Data

Eliminating unnecessary variables: Starting with N / A variables. Next, we proceed to review the variables that have zero or approximately zero variance. Now that we have finished eliminating the unnecessary variables, we can divide the training set into a validation and substraining set. Preparing the data for prediction by splitting the training data into 70% as train data and 30% as test data.

```
set.seed(31824)
traincsv <- traindata[,colMeans(is.na(traindata)) < .9] #removing NA columns
traincsv <- traincsv[,-c(1:7)] #removing metadata which is irrelevant to the outcome

nzv <- nearZeroVar(traincsv)
traincsv <- traincsv[,-nzv]
dim(traincsv)
```

```
## [1] 19622    53
```

```
inTrain <- createDataPartition(y=traincsv$classe, p=0.7, list=F)
train <- traincsv[inTrain,]
valid <- traincsv[-inTrain,]
```

We will do a similar situation for the test data.

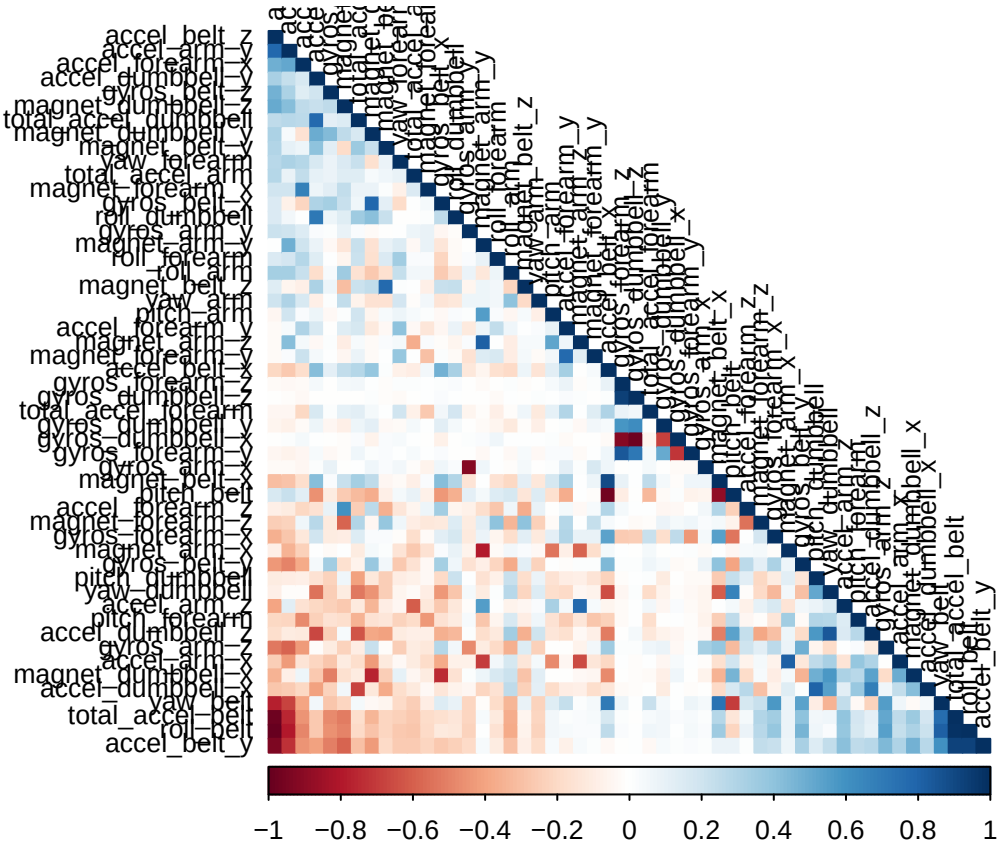
```
set.seed(31824)
testcsv<-testdata[,colMeans(is.na(testdata)) < .9]
testcsv<-testcsv[,-c(1:7)]
```

With the cleaning process above, the number of variables for the analysis has been reduced to 53 only.

Correlation Analysis

A correlation among variables is analysed before proceeding to the modeling procedures.

```
corMatrix <- cor(traincsv[, -53])
corrplot(corMatrix, order = "FPC", method = "color", type = "lower",
         tl.cex = 0.8, tl.col = rgb(0, 0, 0))
```



To obtain the names of the variables we do the following: We use the "findCorrelation" function to search for highly correlated attributes with a cut off equal to 0.8

```
highlyCorrelated = findCorrelation(corMatrix, cutoff=0.8)
```

We then obtain the names of highly correlated attributes:

```
names(traindata)[highlyCorrelated]
```

```
##  [1] "yaw_belt"             "X"                    "pitch_belt"
##  [4] "var_yaw_belt"         "roll_belt"            "user_name"
##  [7] "min_roll_belt"        "avg_yaw_belt"         "amplitude_pitch_belt"
## [10] "magnet_belt_z"        "avg_pitch_belt"       "var_pitch_belt"
## [13] "max_roll_belt"
```

Creating and Testing the Models

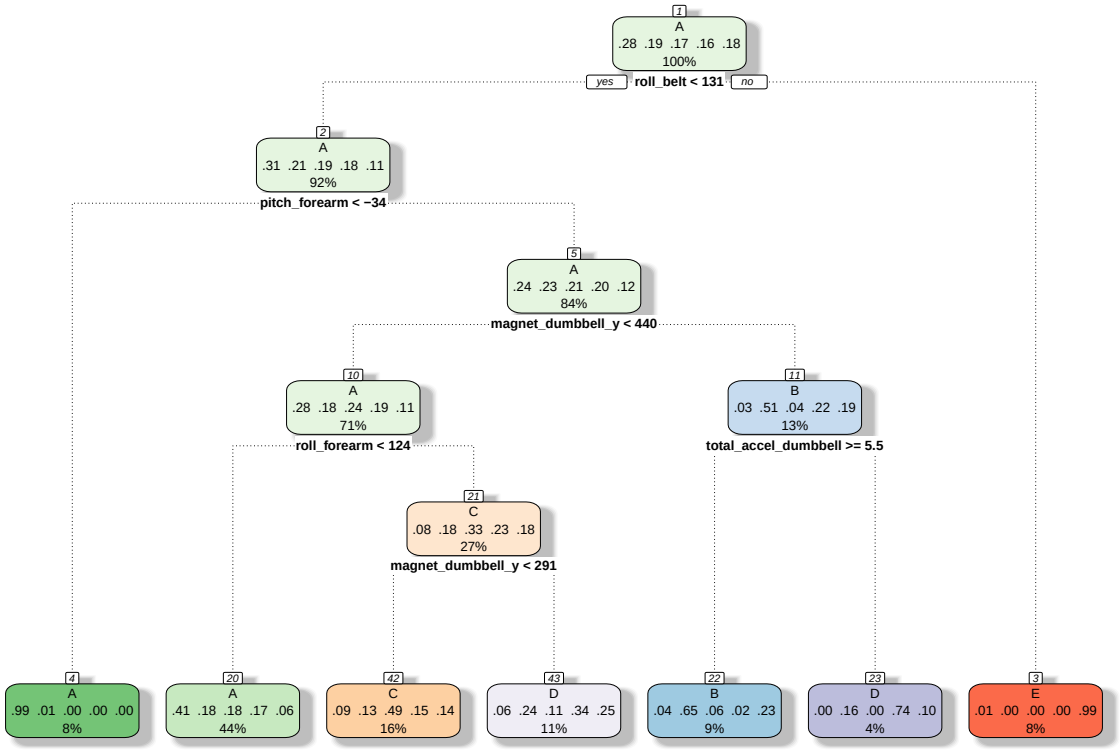
Here we will test a few popular models including: Decision Trees, Random Forest, Gradient Boosted Trees, and SVM. This is probably more than we will need to test, but just out of curiosity and good practice we will run them for comparison.

Set up control for training to use 3-fold cross validation.

```
control <- trainControl(method="cv", number=3, verboseIter=F)
```

Model 1: Decision Tree

```
mod_trees <- train(classe~., data=traincsv, method="rpart", trControl = control, tuneLength = 5)
fancyRpartPlot(mod_trees$finalModel)
```



Rattle 2020−oct.−14 22:49:37 Rob

Prediction:

```
pred_trees <- predict(mod_trees, valid)
cmtrees <- confusionMatrix(pred_trees, factor(valid$classe))
cmtrees
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1530  486  494  418  156
##          B   23  342   34   13  139
##          C   74  116  427  128  125
##          D   41  195   71  405  192
##          E    6    0    0    0  470
##
## Overall Statistics
##
##                Accuracy : 0.5393
##                  95% CI : (0.5265, 0.5521)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3994
```

```
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9140  0.30026  0.41618  0.42012  0.43438
## Specificity            0.6310  0.95596  0.90883  0.89860  0.99875
## Pos Pred Value         0.4961  0.62069  0.49080  0.44801  0.98739
## Neg Pred Value         0.9486  0.85058  0.88056  0.88777  0.88686
## Prevalence             0.2845  0.19354  0.17434  0.16381  0.18386
## Detection Rate         0.2600  0.05811  0.07256  0.06882  0.07986
## Detection Prevalence   0.5240  0.09363  0.14783  0.15361  0.08088
## Balanced Accuracy      0.7725  0.62811  0.66250  0.65936  0.71657
```

Model 2: Random Forest

```
#Random Forest

model_rf <- train(classe~., data=traincsv, method="rf", trControl = control)

pred_rf <- predict(model_rf, valid)
cmrf <- confusionMatrix(pred_rf, factor(valid$classe))
cmrf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    0    0    0    0
##          B    0 1139    0    0    0
##          C    0    0 1026    0    0
##          D    0    0    0  964    0
##          E    0    0    0    0 1082
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9994, 1)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  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   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
```

```
## Detection Rate         0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

Model 3: Gradient Boosted Trees

```
# Gradient Boosted Trees
set.seed(12345)
controlgbm <- trainControl(method = "repeatedcv", number = 5, repeats = 1)
mod_gbm <- train(classe~., data=traincsv, method="gbm", trControl = controlgbm, verbose = F)

pred_gbm <- predict(mod_gbm, valid)
cmgbm <- confusionMatrix(pred_gbm, factor(valid$classe))
cmgbm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1660   26    0    1    0
##          B   12 1092   26    1    5
##          C    1   21  984   24   12
##          D    1    0   13  931   15
##          E    0    0    3    7 1050
##
## Overall Statistics
##
##                Accuracy : 0.9715
##                  95% CI : (0.9669, 0.9756)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9639
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9916   0.9587   0.9591   0.9658   0.9704
## Specificity            0.9936   0.9907   0.9881   0.9941   0.9979
## Pos Pred Value         0.9840   0.9613   0.9443   0.9698   0.9906
## Neg Pred Value         0.9967   0.9901   0.9913   0.9933   0.9934
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2821   0.1856   0.1672   0.1582   0.1784
## Detection Prevalence   0.2867   0.1930   0.1771   0.1631   0.1801
## Balanced Accuracy      0.9926   0.9747   0.9736   0.9799   0.9842
```

Model 4: SVM

```
# SVM
mod_svm <- train(classe~., data=train, method="svmLinear", trControl = control, tuneLength = 5, verbose

pred_svm <- predict(mod_svm, valid)
```

```
cmsvm <- confusionMatrix(pred_svm, factor(valid$classe))
cmsvm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1534  151   84   67   47
##          B   39  807   98   38  155
##          C   41   75  791  110   68
##          D   51   25   28  703   70
##          E    9   81   25   46  742
##
## Overall Statistics
##
##                Accuracy : 0.7777
##                  95% CI : (0.7669, 0.7883)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7175
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9164   0.7085   0.7710   0.7293   0.6858
## Specificity            0.9171   0.9305   0.9395   0.9646   0.9665
## Pos Pred Value         0.8147   0.7098   0.7290   0.8016   0.8217
## Neg Pred Value         0.9650   0.9301   0.9510   0.9479   0.9318
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2607   0.1371   0.1344   0.1195   0.1261
## Detection Prevalence   0.3200   0.1932   0.1844   0.1490   0.1534
## Balanced Accuracy      0.9167   0.8195   0.8552   0.8469   0.8261
```

Applying the best model to the validation data

By comparing the accuracy rate values of the three models, it is clear the the "Random Forest" model is the winner. So will use it on the validation data. Applying the Selected Model to the Test Data

The accuracy of the 4 regression modeling methods above are:

```
Random Forest : 1.0000
Decision Tree : 0.5393
GBM : 0.9715
SVM: 0.7777
```

In that case, the Random Forest model will be applied to predict the 20 quiz results (testing dataset) as shown below.

```
Results <- predict(model_rf, newdata=testcsv)
Results
```

```
##  [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
```