Practical Machine Learning - Course Project

gustavohrp

24 de dezembro de 2015

# Background 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 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).

# Data Source

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://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.

# Project Goal

The goal of your project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

1. Your submission should consist of a link to a Github repo with your R markdown and compiled HTML file describing your analysis. Please constrain the text of the writeup to < 2000 words and the number of figures to be less than 5. It will make it easier for the graders if you submit a repo with a gh-pages branch so the HTML page can be viewed online (and you always want to make it easy on graders :-).
2. You should also apply your machine learning algorithm to the 20 test cases available in the test data above. Please submit your predictions in appropriate format to the programming assignment for automated grading. See the programming assignment for additional details.

# Loading Library and Getting Data

The following Libraries were used for this project, which you should install - if not done yet - and load on your working environment.

# Libraries used for the data processing  
library(caret)

## Loading required package: lattice  
## Loading required package: ggplot2

library(rpart)  
library(rpart.plot)  
library(RColorBrewer)  
library(rattle)

## Warning: package 'rattle' was built under R version 3.2.2

## Rattle: A free graphical interface for data mining with R.  
## Version 4.0.5 Copyright (c) 2006-2015 Togaware Pty Ltd.  
## Type 'rattle()' to shake, rattle, and roll your data.

library(randomForest)

## randomForest 4.6-12  
## Type rfNews() to see new features/changes/bug fixes.

library(knitr)

Finally, load the same seed with the following line of code:

The training data set can be found on the following URL:

rm(list = ls())  
if (!file.exists("pml-training.csv")) {  
 download.file("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv", destfile = "pml-training.csv")  
}  
if (!file.exists("pml-testing.csv")) {  
 download.file("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv", destfile = "pml-testing.csv")  
}  
training <- read.csv("pml-training.csv", na.strings=c("NA","#DIV/0!",""))  
testing <- read.csv("pml-testing.csv", na.strings=c("NA","#DIV/0!",""))

In order to reproduce the same results, you need a certain set of packages, as well as setting a pseudo random seed equal to the one I used.

# Cleaning up the data and Bootstrap

Here, I remove columns full of NAs and remove features that are not in the testing set. The features containing NAs are the variance, mean and stddev within each window for each feature. Since the testing dataset has no time-dependence, these values are useless and can be disregarded. I also remove the first 7 features since they are related to the time-series or are not numeric.

inTrain <- createDataPartition(training$classe, p=0.6, list=FALSE)  
myTraining <- training[inTrain, ]  
myTesting <- training[-inTrain, ]  
dim(myTraining); dim(myTesting)

## [1] 11776 160

## [1] 7846 160

I am now going to reduce the number of features by removing variables with nearly zero variance, variables that are almost always NA, and variables that don’t make intuitive sense for prediction. Note that I decide which ones to remove by analyzing ptrain1, and perform the identical removals on ptrain2:

nzv <- nearZeroVar(myTraining, saveMetrics=TRUE)  
myTraining <- myTraining[,nzv$nzv==FALSE]  
  
nzv<- nearZeroVar(myTesting,saveMetrics=TRUE)  
myTesting <- myTesting[,nzv$nzv==FALSE]  
  
myTraining <- myTraining[c(-1)]  
trainingV3 <- myTraining  
for(i in 1:length(myTraining)) {  
 if( sum( is.na( myTraining[, i] ) ) /nrow(myTraining) >= .7) {  
 for(j in 1:length(trainingV3)) {  
 if( length( grep(names(myTraining[i]), names(trainingV3)[j]) ) == 1) {  
 trainingV3 <- trainingV3[ , -j]  
 }   
 }   
 }  
}  
  
# Set back to the original variable name  
myTraining <- trainingV3  
rm(trainingV3)

clean1 <- colnames(myTraining)  
clean2 <- colnames(myTraining[, -58]) # remove the classe column  
myTesting <- myTesting[clean1] # allow only variables in myTesting that are also in myTraining  
testing <- testing[clean2] # allow only variables in testing that are also in myTraining  
  
dim(myTesting)

## [1] 7846 58

dim(testing)

## [1] 20 57

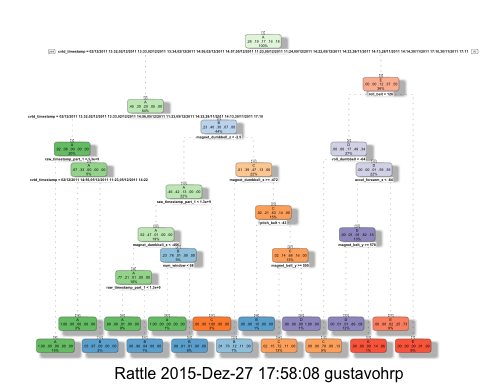
for (i in 1:length(testing) ) {  
 for(j in 1:length(myTraining)) {  
 if( length( grep(names(myTraining[i]), names(testing)[j]) ) == 1) {  
 class(testing[j]) <- class(myTraining[i])  
 }   
 }   
}  
  
# To get the same class between testing and myTraining  
testing <- rbind(myTraining[2, -58] , testing)  
testing <- testing[-1,]

# Model Building

I decided to start with a Random Forest model, to see if it would have acceptable performance. I fit the model on ptrain1, and instruct the “train” function to use 3-fold cross-validation to select optimal tuning parameters for the model.

I see that it decided to use 500 trees and try 27 variables at each split. Now, I use the fitted model to predict the label (“classe”) in ptrain2, and show the confusion matrix to compare the predicted versus the actual labels:

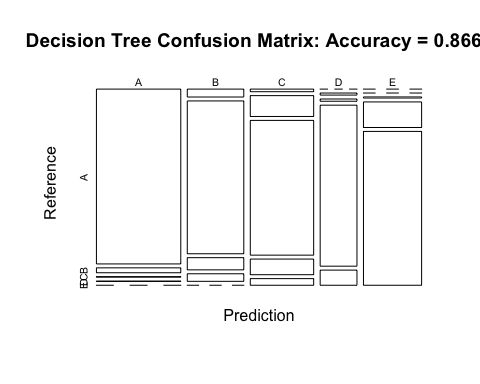
set.seed(12345)  
modFitA1 <- rpart(classe ~ ., data=myTraining, method="class")  
fancyRpartPlot(modFitA1)



predictionsA1 <- predict(modFitA1, myTesting, type = "class")  
cmtree <- confusionMatrix(predictionsA1, myTesting$classe)  
cmtree

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 2143 60 6 3 0  
## B 65 1255 100 62 0  
## C 24 192 1241 144 61  
## D 0 11 12 861 81  
## E 0 0 9 216 1300  
##   
## Overall Statistics  
##   
## Accuracy : 0.8667   
## 95% CI : (0.859, 0.8741)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8313   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.9601 0.8267 0.9072 0.6695 0.9015  
## Specificity 0.9877 0.9641 0.9350 0.9841 0.9649  
## Pos Pred Value 0.9688 0.8468 0.7467 0.8922 0.8525  
## Neg Pred Value 0.9842 0.9587 0.9795 0.9382 0.9775  
## Prevalence 0.2845 0.1935 0.1744 0.1639 0.1838  
## Detection Rate 0.2731 0.1600 0.1582 0.1097 0.1657  
## Detection Prevalence 0.2819 0.1889 0.2118 0.1230 0.1944  
## Balanced Accuracy 0.9739 0.8954 0.9211 0.8268 0.9332

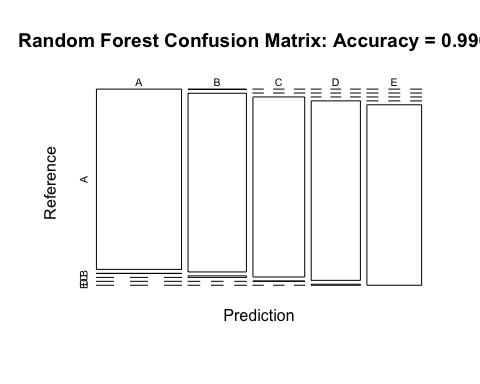
plot(cmtree$table, col = cmtree$byClass, main = paste("Decision Tree Confusion Matrix: Accuracy =", round(cmtree$overall['Accuracy'], 4)))



set.seed(12345)  
modFitB1 <- randomForest(classe ~ ., data=myTraining)  
predictionB1 <- predict(modFitB1, myTesting, type = "class")  
cmrf <- confusionMatrix(predictionB1, myTesting$classe)  
cmrf

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 2230 2 0 0 0  
## B 2 1516 13 0 0  
## C 0 0 1355 3 0  
## D 0 0 0 1283 7  
## E 0 0 0 0 1435  
##   
## Overall Statistics  
##   
## Accuracy : 0.9966   
## 95% CI : (0.995, 0.9977)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9956   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.9991 0.9987 0.9905 0.9977 0.9951  
## Specificity 0.9996 0.9976 0.9995 0.9989 1.0000  
## Pos Pred Value 0.9991 0.9902 0.9978 0.9946 1.0000  
## Neg Pred Value 0.9996 0.9997 0.9980 0.9995 0.9989  
## Prevalence 0.2845 0.1935 0.1744 0.1639 0.1838  
## Detection Rate 0.2842 0.1932 0.1727 0.1635 0.1829  
## Detection Prevalence 0.2845 0.1951 0.1731 0.1644 0.1829  
## Balanced Accuracy 0.9994 0.9982 0.9950 0.9983 0.9976

plot(cmrf$table, col = cmtree$byClass, main = paste("Random Forest Confusion Matrix: Accuracy =", round(cmrf$overall['Accuracy'], 4)))



The accuracy is 99.8%, thus my predicted accuracy for the out-of-sample error is 0.2%.

This is an excellent result, so rather than trying additional algorithms, I will use Random Forests to predict on the test set.

# Re-training the Selected Model

Before predicting on the test set, it is important to train the model on the full training set (ptrain), rather than using a model trained on a reduced training set (ptrain1), in order to produce the most accurate predictions. Therefore, I now repeat everything I did above on ptrain and ptest:

predictionB2 <- predict(modFitB1, testing, type = "class")  
predictionB2

## 2 3 41 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21   
## 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

# Write the results to a text file for submission  
pml\_write\_files = function(x){  
 n = length(x)  
 for(i in 1:n){  
 filename = paste0("problem\_id\_",i,".txt")  
 write.table(x[i],file=filename,quote=FALSE,row.names=FALSE,col.names=FALSE)  
 }  
}  
  
# pml\_write\_files(predictionB2)