Project Assignment of Practical Machine Learning Course

Tran Ho Thanh Dong

Monday, April 18, 2015

## 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 â 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).

## Purpose

The goal of the project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set. The other variables are used to predict with. Some various models are used for predicting and Cross validation is used in the process. The accuracy and out of sample error is also analyzed. The report also present predicted result as applying the models to 20 different test cases.

## Load Libraries

library(caret)  
library(rpart)  
library(rpart.plot)  
library(rattle)  
library(randomForest)

## Data Processing

### Getting Data

* Due to the conenction problem with loading data from web for some kind of network, I used the method of loading data files from **working directory** into **trainfile** and **testfile** data sets

trainfile<-read.csv("pml-training.csv")  
testfile<-read.csv("pml-testing.csv")  
dim(trainfile)

## [1] 19622 160

dim(testfile)

## [1] 20 160

#trainfile <- read.csv('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv')  
#testfile <- read.csv('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv')

**The total predictors are 160**

### Cleaning data

* All predictors which are calculated based on other predictors such as **max, min, std, avg, var**... should be removed

col\_names<-names(trainfile)  
grepindex<-grep("max\_|min\_|avg\_|stddev\_|var\_|amplitude\_",col\_names,value=FALSE)  
train\_raw<-trainfile[-grepindex]  
test\_raw<-testfile[-grepindex]

* The first 7 columns which are not predictors to predict the **"classe"** variable should be removed

head(train\_raw[,1:7])

## X user\_name raw\_timestamp\_part\_1 raw\_timestamp\_part\_2 cvtd\_timestamp  
## 1 1 carlitos 1323084231 788290 05/12/2011 11:23  
## 2 2 carlitos 1323084231 808298 05/12/2011 11:23  
## 3 3 carlitos 1323084231 820366 05/12/2011 11:23  
## 4 4 carlitos 1323084232 120339 05/12/2011 11:23  
## 5 5 carlitos 1323084232 196328 05/12/2011 11:23  
## 6 6 carlitos 1323084232 304277 05/12/2011 11:23  
## new\_window num\_window  
## 1 no 11  
## 2 no 11  
## 3 no 11  
## 4 no 12  
## 5 no 12  
## 6 no 12

train\_raw<-train\_raw[,-c(1:7)]  
test\_raw<-test\_raw[,-c(1:7)]

##### The total predictors now are 77

* There are still predictors that have very few unique values relative to the number of samples and the ratio of the frequency of the most common value to the frequency of the second most common value is large. The function **nearZeroVar** is used to locate those predictors and finally, they should be removed out of **the train\_raw/test\_raw sets**

NZVindex <- nearZeroVar(train\_raw, saveMetrics=FALSE)  
training<-train\_raw[-NZVindex]  
testing<-test\_raw[-NZVindex]  
dim(training)

## [1] 19622 53

The results are data sets with **53** predictors which are:

names(training)

## [1] "roll\_belt" "pitch\_belt" "yaw\_belt"   
## [4] "total\_accel\_belt" "gyros\_belt\_x" "gyros\_belt\_y"   
## [7] "gyros\_belt\_z" "accel\_belt\_x" "accel\_belt\_y"   
## [10] "accel\_belt\_z" "magnet\_belt\_x" "magnet\_belt\_y"   
## [13] "magnet\_belt\_z" "roll\_arm" "pitch\_arm"   
## [16] "yaw\_arm" "total\_accel\_arm" "gyros\_arm\_x"   
## [19] "gyros\_arm\_y" "gyros\_arm\_z" "accel\_arm\_x"   
## [22] "accel\_arm\_y" "accel\_arm\_z" "magnet\_arm\_x"   
## [25] "magnet\_arm\_y" "magnet\_arm\_z" "roll\_dumbbell"   
## [28] "pitch\_dumbbell" "yaw\_dumbbell" "total\_accel\_dumbbell"  
## [31] "gyros\_dumbbell\_x" "gyros\_dumbbell\_y" "gyros\_dumbbell\_z"   
## [34] "accel\_dumbbell\_x" "accel\_dumbbell\_y" "accel\_dumbbell\_z"   
## [37] "magnet\_dumbbell\_x" "magnet\_dumbbell\_y" "magnet\_dumbbell\_z"   
## [40] "roll\_forearm" "pitch\_forearm" "yaw\_forearm"   
## [43] "total\_accel\_forearm" "gyros\_forearm\_x" "gyros\_forearm\_y"   
## [46] "gyros\_forearm\_z" "accel\_forearm\_x" "accel\_forearm\_y"   
## [49] "accel\_forearm\_z" "magnet\_forearm\_x" "magnet\_forearm\_y"   
## [52] "magnet\_forearm\_z" "classe"

## K-Fold Cross-Validation

### Number of folds to partition the training set

fold<-10

* **The training set** is very big with dimension of **19622** x **53**. In order to reduce variability. instead of performing the **Machine Learning Algorithm** on the entire training set, it is randomly partitioned into 10 equal size subsamples.
* Multiple round of cross-validation are performed using each of above partitions, and the validation results are averaged over the rounds

set.seed(1967)  
folds <- createFolds(training$classe, k = fold, list = TRUE, returnTrain = FALSE)  
  
sample\_list<-list()  
  
for(i in 1:fold){  
 sample\_list[[i]]<-training[folds[[i]],]  
}

### Partritioning the folds of training set

* The folds are partitioned into **60%** for training sets and **40%** for testing sets
* Store the training and testing sets in list variables called **training\_list** and **testing\_list**

training\_list<-list()  
testing\_list<-list()  
  
for(i in 1:fold){  
 set.seed(1967)  
 index <- createDataPartition(y=sample\_list[[i]]$classe, p=0.6, list=FALSE)  
 training\_list[[i]] <- sample\_list[[i]][index, ]  
 testing\_list[[i]] <- sample\_list[[i]][-index, ]  
}

## Machine Learning Algorithm

* The Machine Learning Algorithms **Decision Tree** and **Random Forest** which are done both with and without **preprocessing** are applied to the training samples in the **training\_list**. The fit models are stored either in a list variable called **fit\_method\_list** or **fit\_preprocessing\_method\_list** if the preprocessing is done in the model
* The results of predicting on the testing samples the **testing\_list** are stored either in a list variable called **predict\_method\_list** or **predict\_preprocessing\_method\_list** if the preprocessing is done in the model
* The confusionMatrix of predicting on the testing samples the **testing\_list** are stored either in a list variable called **confusion\_method\_list** or **confusion\_preprocessing\_method\_list** if the preprocessing is done in the model
* The accuracy values of confusionMatrix for each model are stored either in a vector variable called **accuracy\_method\_list** or **accuracy\_preprocessing\_method\_list** if the preprocessing is done in the model

### Decision Tree

* **Without Preprocessing**

fit\_tree\_list<-list()  
predict\_tree\_list<-list()  
confusion\_tree\_list<-list()  
accuracy\_tree\_list<-numeric()  
  
for(i in 1:fold){  
 set.seed(1967)  
 fit\_tree\_list[[i]] <- train(classe ~ ., data=training\_list[[i]], method="rpart", trControl=trainControl(method = "cv", number = 4))  
 predict\_tree\_list[[i]]<-predict(fit\_tree\_list[[i]], testing\_list[[i]])  
 confusion\_tree\_list[[i]]<-confusionMatrix(predict\_tree\_list[[i]], testing\_list[[i]]$classe)  
 accuracy\_tree\_list[[i]]<-confusion\_tree\_list[[i]]$overall[1]  
}

* **With Preprocessing**

fit\_preprocess\_tree\_list<-list()  
predict\_preprocess\_tree\_list<-list()  
confusion\_preprocess\_tree\_list<-list()  
accuracy\_preprocess\_tree\_list<-numeric()  
  
for(i in 1:fold){  
 set.seed(1967)  
 fit\_preprocess\_tree\_list[[i]] <- train(classe ~ ., data=training\_list[[i]], method="rpart", preProcess=c("center", "scale"), trControl=trainControl(method = "cv", number = 4))  
 predict\_preprocess\_tree\_list[[i]]<-predict(fit\_preprocess\_tree\_list[[i]], testing\_list[[i]])  
 confusion\_preprocess\_tree\_list[[i]]<-confusionMatrix(predict\_preprocess\_tree\_list[[i]], testing\_list[[i]]$classe)  
 accuracy\_preprocess\_tree\_list[[i]]<-confusion\_preprocess\_tree\_list[[i]]$overall[1]  
}

### Random Forest

* **Without Preprocessing**

fit\_rf\_list<-list()  
predict\_rf\_list<-list()  
confusion\_rf\_list<-list()  
accuracy\_rf\_list<-numeric()  
  
for(i in 1:fold){  
 set.seed(1967)  
 fit\_rf\_list[[i]] <- train(classe ~ ., data=training\_list[[i]], method="rf", trControl=trainControl(method = "cv", number = 4))  
 predict\_rf\_list[[i]]<-predict(fit\_rf\_list[[i]], testing\_list[[i]])  
 confusion\_rf\_list[[i]]<-confusionMatrix(predict\_rf\_list[[i]], testing\_list[[i]]$classe)  
 accuracy\_rf\_list[[i]]<-confusion\_rf\_list[[i]]$overall[1]  
}

* **With Preprocessing**

fit\_preprocess\_rf\_list<-list()  
predict\_preprocess\_rf\_list<-list()  
confusion\_preprocess\_rf\_list<-list()  
accuracy\_preprocess\_rf\_list<-numeric()  
  
for(i in 1:fold){  
 set.seed(1967)  
 fit\_preprocess\_rf\_list[[i]] <- train(classe ~ ., data=training\_list[[i]], method="rf", preProcess=c("center", "scale"), trControl=trainControl(method = "cv", number = 4))  
 predict\_preprocess\_rf\_list[[i]]<-predict(fit\_preprocess\_rf\_list[[i]], testing\_list[[i]])  
 confusion\_preprocess\_rf\_list[[i]]<-confusionMatrix(predict\_preprocess\_rf\_list[[i]], testing\_list[[i]]$classe)  
 accuracy\_preprocess\_rf\_list[[i]]<-confusion\_preprocess\_rf\_list[[i]]$overall[1]  
}

#### The accuracy values and its mean values of each model

1- *Decision tree without preprocessing*

options(width = 110)  
accuracy\_tree\_list

## [1] 0.4424552 0.4968072 0.4258312 0.3639847 0.4840358 0.5165816 0.4884910 0.4750958 0.5670498 0.4744898

mean(accuracy\_tree\_list)

## [1] 0.4734822

2- *Decision tree with preprocessing*

options(width = 110)  
accuracy\_preprocess\_tree\_list

## [1] 0.4424552 0.4968072 0.4258312 0.3639847 0.4840358 0.5165816 0.4884910 0.4750958 0.5670498 0.4732143

mean(accuracy\_preprocess\_tree\_list)

## [1] 0.4733547

3- *Random Forest without preprocessing*

options(width = 110)  
accuracy\_rf\_list

## [1] 0.9373402 0.9246488 0.9296675 0.9310345 0.9272031 0.9005102 0.9347826 0.9374202 0.9080460 0.9234694

mean(accuracy\_rf\_list)

## [1] 0.9254122

4- *Random Forest with preprocessing*

options(width = 110)  
accuracy\_preprocess\_rf\_list

## [1] 0.9398977 0.9323116 0.9283887 0.9284802 0.9157088 0.9017857 0.9335038 0.9361430 0.9118774 0.9260204

mean(accuracy\_preprocess\_rf\_list)

## [1] 0.9254117

* The results show that the model built based on algorithm **Random Forest** without preprocessing has the highest mean of accuracy values.
* Thus, our model is built based on the algorithm **Random Forest** without preprocessing

#### The out of sample error of each model

1- *Decision tree without preprocessing*

1-mean(accuracy\_tree\_list)

## [1] 0.5265178

2- *Decision tree with preprocessing*

1-mean(accuracy\_preprocess\_tree\_list)

## [1] 0.5266453

3- *Random Forest without preprocessing*

1-mean(accuracy\_rf\_list)

## [1] 0.07458776

4- *Random Forest with preprocessing*

1-mean(accuracy\_preprocess\_rf\_list)

## [1] 0.07458825

## Conclussion

* The model which is built based on the agorithm Random Forest without preprocessing has the highest accuracy. It means the out of sampe error of this model is the lowest. Thus, the out of sample error should be **0.0745878**

## Submmision

* Apply the models built for 10 folds based on algorithm **Random Forest** without preprocessing to the 20 test cases available in the test data provided

predict<-list()  
for(i in 1:fold){  
 predict[[i]]<-predict(fit\_rf\_list[[i]], testing)  
}

* The results of predicting **classe** from 10 above models are

options(width = 100)  
for(i in 1:fold){  
 print(as.character(predict[[i]]))   
}

## [1] "B" "A" "B" "A" "A" "E" "D" "D" "A" "A" "C" "C" "B" "A" "E" "E" "A" "B" "A" "B"  
## [1] "B" "A" "B" "A" "A" "E" "D" "D" "A" "A" "C" "C" "B" "A" "E" "E" "A" "D" "B" "B"  
## [1] "B" "A" "A" "A" "A" "E" "D" "D" "A" "A" "B" "C" "B" "A" "E" "E" "A" "B" "B" "B"  
## [1] "C" "A" "B" "A" "A" "E" "D" "D" "A" "A" "B" "C" "B" "A" "E" "E" "A" "B" "B" "B"  
## [1] "B" "A" "A" "A" "A" "E" "D" "D" "A" "A" "B" "C" "B" "A" "E" "E" "A" "D" "B" "B"  
## [1] "C" "A" "C" "A" "A" "E" "D" "A" "A" "A" "B" "C" "B" "A" "E" "E" "A" "B" "B" "B"  
## [1] "B" "A" "B" "A" "A" "C" "D" "D" "A" "A" "B" "C" "B" "A" "E" "E" "A" "B" "B" "B"  
## [1] "C" "A" "C" "A" "A" "E" "D" "D" "A" "A" "B" "C" "B" "A" "E" "E" "A" "D" "A" "B"  
## [1] "C" "A" "B" "A" "A" "E" "D" "B" "A" "A" "B" "C" "B" "A" "E" "E" "A" "B" "B" "B"  
## [1] "B" "A" "C" "A" "A" "E" "D" "B" "A" "A" "C" "C" "B" "A" "E" "E" "A" "B" "B" "B"

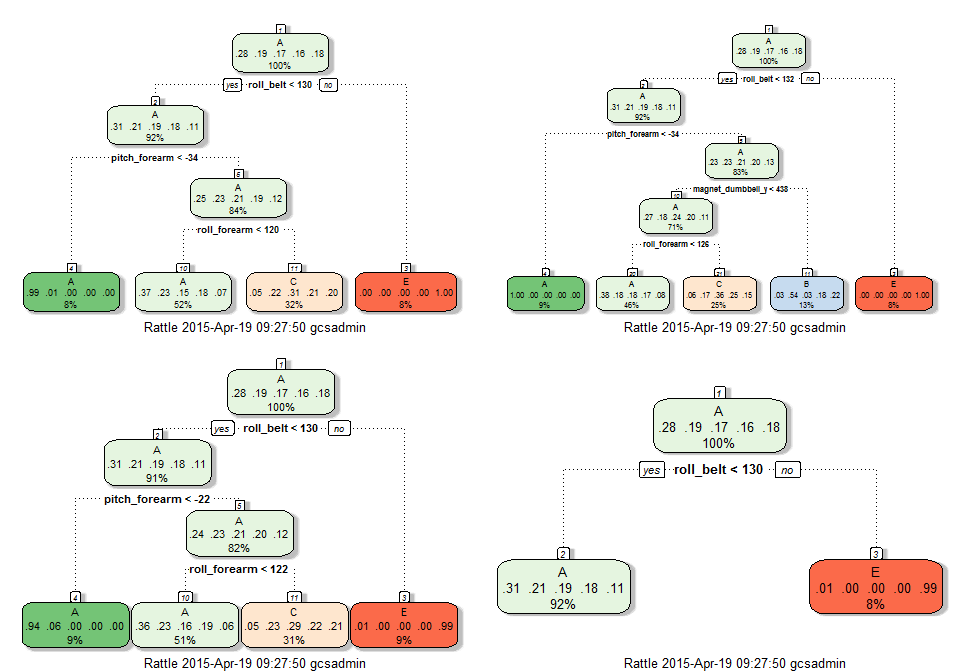
* Because, We have 02 tries for each letter in predicted **classe**, 02 letters that appear the most at each position predicted by the 10 above models should be chosen for submission. Thus, the result should be:

"**B or C**"" "**A**" "**B or C**" "**A**" "**A**" "**E**" "**D**" "**D or B**" "**A**" "**A**" "**C or B**" "**C**" "**B**" "**A**" "**E**" "**E**" "**A**" "**B or D**" "**A or B**" "**B**"

## Appendix: 04 plots

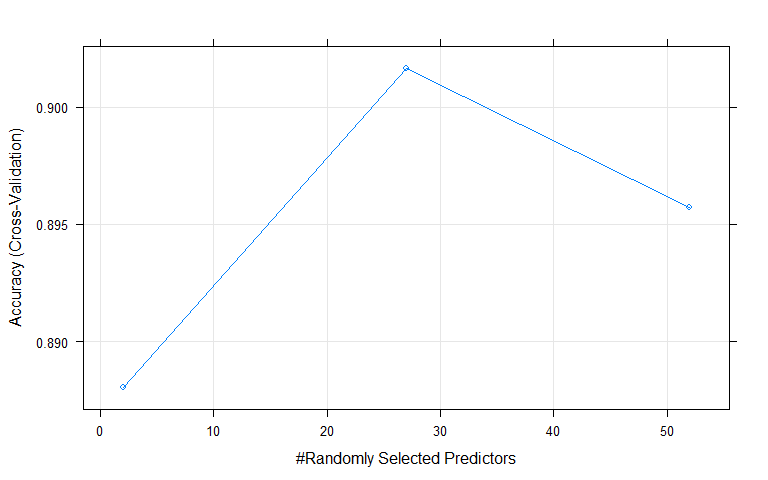
* 01 PanelPlot for Classification Tree of models for the first four folds of the training set

par(mfrow=c(2,2))  
for(i in 1:4)  
 fancyRpartPlot(fit\_tree\_list[[i]]$finalModel)

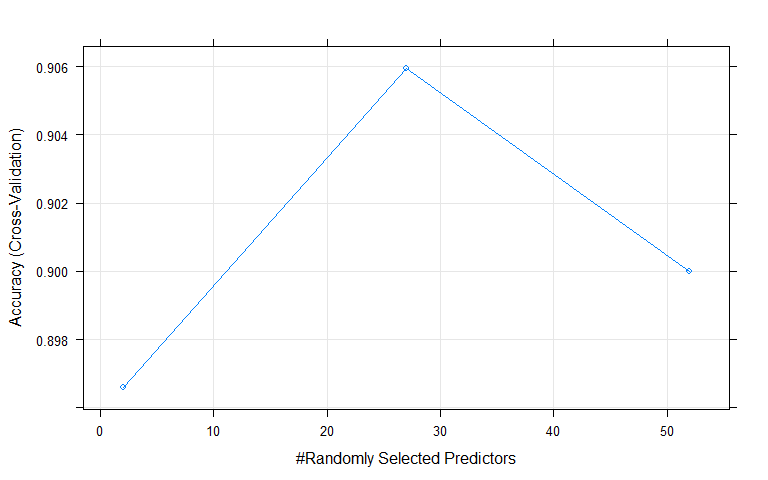


* 3 plots for models built based on algorithm Random Forest for the first four folds of the training set

plot(fit\_rf\_list[[1]],log="y")



plot(fit\_rf\_list[[2]],log="y")



plot(fit\_rf\_list[[3]],log="y")

