Practical MachineLearning: Final Project

Rolando P. Hong Enriquez

Thursday, April 23, 2015

### Introduction

In this report an estrategy to implement several machine learning (ML) techniques is applied to solve a classification problem in a data base containing measures (mostly from accelerometers located in several parts of the body) aimed to monitor the quality of human physical activities. Detailed information the data sets used in this report can be obtained directly from the website <http://groupware.les.inf.puc-rio.br/har>. The corresponding references from the authors is:

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.

Specifically, in this report using ML technoques we will try to predict the way in which the human activities (exercices) were performed. This information in the training sets will be saved in the variable "classe", which will therefore be the response variable in the study.

## Getting and preprocessing data.

The original training set (containing the variable "classe") and test set (containing the 20 test set cases to be delivered) were downloaded from:

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

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

and they were loaded into R homogenizing at the same type the 'NA' values:

training<-read.csv("pml-training.csv",na.strings=c("NA","#DIV/0!",""))  
testing<-read.csv("pml-testing.csv",na.strings=c("NA","#DIV/0!",""))

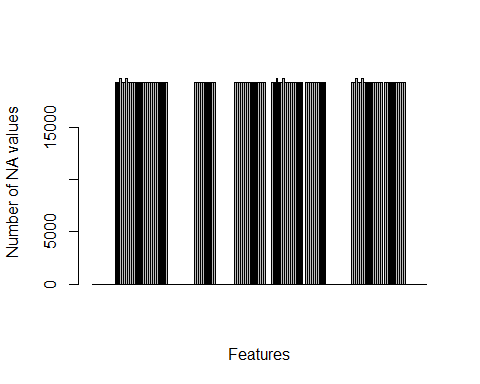
As we can see, the original training set has 160 features and 19622 training examples:

dim(training)

## [1] 19622 160

Next we explore the training data set to see how many 'NA' values are present on each feature:

tmp<-dim(training)  
d=tmp[2]  
colNA<-0  
for (i in 1:d){  
 s<-sum(is.na(training[,i]))  
 colNA<-c(colNA,s)  
}  
colNA<-colNA[2:length(colNA)]  
barplot(colNA,xlab="Features",ylab="Number of NA values")



From this bar plot we calsee that there are many features with a large number of 'NA' values, therefore an initial preprocesing step would be to just get rid of these features as they will not have a positive impact on a prediction. We also eliminate the first 7 features which are uninformative for prediction.

myfeatures<-NULL  
nm<-names(training)  
for (i in 1:d){  
 if (colNA[i]==0){  
 myfeatures<-c(myfeatures,i)  
 }  
}  
train<-training[myfeatures]  
test<-testing[myfeatures]  
train<-train[8:length(myfeatures)]  
test<-test[8:length(myfeatures)]  
dim(train)

## [1] 19622 53

As we see, the new (cleaned) training set named 'train' has only 53 features. (a total of 107 features were eliminated).

## Creating the model

Before creating the model we first load all the libraries that will be needed and set the seed to ensure the reproducibility on the results by other analysts.

library(caret)

## Warning: package 'caret' was built under R version 3.1.3

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

library(rpart)

## Warning: package 'rpart' was built under R version 3.1.3

library(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 3.1.3

library(rattle)

## Warning: package 'rattle' was built under R version 3.1.3

## Rattle: A free graphical interface for data mining with R.  
## Version 3.4.1 Copyright (c) 2006-2014 Togaware Pty Ltd.  
## Type 'rattle()' to shake, rattle, and roll your data.

library(randomForest)

## Warning: package 'randomForest' was built under R version 3.1.3

## randomForest 4.6-10  
## Type rfNews() to see new features/changes/bug fixes.

set.seed(1000)

A first step in creating the model is the partitioning of the data:

inTraining <- createDataPartition(train$classe, p = .75, list = FALSE)  
TRN <- train[ inTraining,]  
TST <- train[-inTraining,]  
dim(TRN)

## [1] 14718 53

# Classification Tree

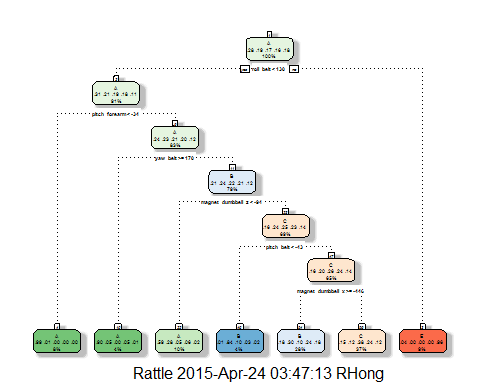
Next we define some fine tunning to be used in the model. Specifically we substitute the default cross validation by a k-fold cross validation with K=10 (as we saw the training data set to be finaly used, 'TRN', still has more than 14000 values which justifies our choice of k). Subsequently wefir a model using as method a classification tree ('rpart').  
.

fitControl <- trainControl(method = "repeatedcv",number = 10,repeats = 10)  
modelTree<-train(classe ~ .,data=TRN,method="rpart",trControl=fitControl)  
modelTree

## CART   
##   
## 14718 samples  
## 52 predictor  
## 5 classes: 'A', 'B', 'C', 'D', 'E'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 10 times)   
##   
## Summary of sample sizes: 13246, 13245, 13248, 13246, 13247, 13247, ...   
##   
## Resampling results across tuning parameters:  
##   
## cp Accuracy Kappa Accuracy SD Kappa SD   
## 0.04144118 0.4983040 0.34926073 0.05545189 0.09275295  
## 0.04260894 0.4412010 0.25299677 0.07096229 0.12128892  
## 0.11696573 0.3175842 0.05094428 0.03930267 0.06025007  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was cp = 0.04144118.

As we see, with this strategy we obtained a trained model with a not very good accuracy (0.49). A visualization of the final model can be obsserved in the next figure:

fancyRpartPlot(modelTree$finalModel)



We can also see the performace of this classification tree model on new test data. It is worth noting tha the use of extensive k-fold cross validation estrategy in the training set allowed us to avoid overfitting, indeed the out of the sample (generalization) error in this new data is remarkably good.

library(caret)  
modelTree.prediction<-predict(modelTree,TST)  
confusionMatrix(modelTree.prediction,TST$classe)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 837 144 21 37 11  
## B 252 586 161 347 252  
## C 283 219 673 420 222  
## D 0 0 0 0 0  
## E 23 0 0 0 416  
##   
## Overall Statistics  
##   
## Accuracy : 0.5122   
## 95% CI : (0.4981, 0.5263)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.3865   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.6000 0.6175 0.7871 0.0000 0.46171  
## Specificity 0.9393 0.7441 0.7175 1.0000 0.99425  
## Pos Pred Value 0.7971 0.3667 0.3704 NaN 0.94761  
## Neg Pred Value 0.8552 0.8902 0.9410 0.8361 0.89138  
## Prevalence 0.2845 0.1935 0.1743 0.1639 0.18373  
## Detection Rate 0.1707 0.1195 0.1372 0.0000 0.08483  
## Detection Prevalence 0.2141 0.3259 0.3705 0.0000 0.08952  
## Balanced Accuracy 0.7696 0.6808 0.7523 0.5000 0.72798

# Random Forest

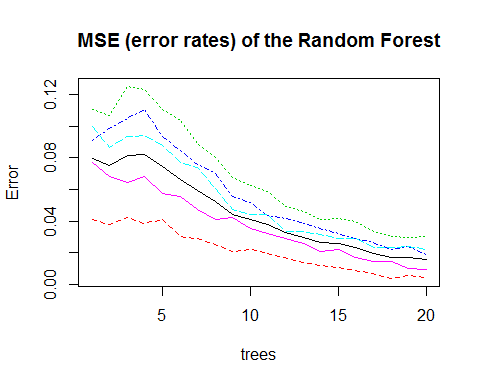
Still, with this poor results we made a change to a more powerful approach: random forest.

library(randomForest)  
modelRF<-randomForest(classe ~ ., TRN,ntree=20,norm.votes=FALSE)  
modelRF

##   
## Call:  
## randomForest(formula = classe ~ ., data = TRN, ntree = 20, norm.votes = FALSE)   
## Type of random forest: classification  
## Number of trees: 20  
## No. of variables tried at each split: 7  
##   
## OOB estimate of error rate: 1.57%  
## Confusion matrix:  
## A B C D E class.error  
## A 4165 7 4 8 0 0.004541109  
## B 35 2762 33 9 9 0.030196629  
## C 2 24 2518 19 3 0.018706157  
## D 5 5 39 2359 4 0.021973466  
## E 2 10 3 10 2681 0.009238729

A plot of the training error rates for the generated random forest shows that a forest with 20 trees is good enough as the errors are minimized.

plot(modelRF,main="MSE (error rates) of the Random Forest")



In fact the predictions with the random forest have a very high quality:

library(caret)  
modelRF.prediction<-predict(modelRF,TST)  
confusionMatrix(modelRF.prediction,TST$classe)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 1391 0 1 0 0  
## B 3 947 7 0 0  
## C 0 2 842 5 0  
## D 0 0 5 797 4  
## E 1 0 0 2 897  
##   
## Overall Statistics  
##   
## Accuracy : 0.9939   
## 95% CI : (0.9913, 0.9959)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9923   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.9971 0.9979 0.9848 0.9913 0.9956  
## Specificity 0.9997 0.9975 0.9983 0.9978 0.9993  
## Pos Pred Value 0.9993 0.9896 0.9918 0.9888 0.9967  
## Neg Pred Value 0.9989 0.9995 0.9968 0.9983 0.9990  
## Prevalence 0.2845 0.1935 0.1743 0.1639 0.1837  
## Detection Rate 0.2836 0.1931 0.1717 0.1625 0.1829  
## Detection Prevalence 0.2838 0.1951 0.1731 0.1644 0.1835  
## Balanced Accuracy 0.9984 0.9977 0.9915 0.9945 0.9974

## Final Evaluation

Now we use the random forest model to make the 20 prediction to be submitted in this exercice. First we can reformat the final test set to have the same structure as the training set used to build the models.

colnames(test)[53]<-"classe"  
test$classe[1]<-"A"  
test$classe[2]<-"B"  
test$classe[3]<-"C"  
test$classe[4]<-"D"  
test$classe[5]<-"E"  
test$classe[6:20]<-"A"  
test[,"classe"]<-as.factor(test[,"classe"])

and we finaly made the predictions:

predT<-predict(modelRF,test)  
predT

## 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