Practical Machine Learning - Final Assignment

Paolo Coraggio

27/12/2019

# Introduction

## Project Outline

The goal of the final assignment is to use data from accelerometers placed on the belt, forearm, arm, and dumbell of 6 participants of an experiment to build a predictive model about the manner in which they did the exercise.

In order to fulfil the goal, the data collected in <http://groupware.les.inf.puc-rio.br/har> were loaded explored and filtered. Then, 4 different predictive models (using Regression Tree, Random Forest, Bagging and Boosting algorithm) build and compared using a similar setting. The model that resulted with higerh accuracy was further improved and, finally, used on the test dataset.

This final report will describe how the model was built 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.

## Exploratory data analysis

In order to build the predective model, the

### Loading the data

The main source of the data is <http://groupware.les.inf.puc-rio.br/har>. The data were downloaded in a local folder and then loaded in two different datasets: dataset for the training and testset for the testing sets

The datasets have the following size:

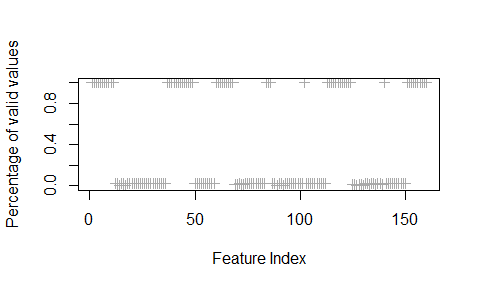
## [1] 19622 160

## [1] 20 160

The dataset data frame consists of datapoint of recorded features and the testset it’s just data points.

### Exploratoring Data Analysis and Feature Extraction

The first step is to check what is the percentage of available data for each feature as the data.frame columns may contain not valid elements. I created a validelements function that, for each data.frame column, count their valid elements (i.e. not empty, NaN or #DIV/0!). I run the function on the dataset data.frame.



The plot shows that data.frame variables contain or valide data or very few (less than of valide data). The features containing less the of data will be discharged.

Moreover, we can further exclude the first 7 features as they containg temporal information that has been chosen not to consider as the analysis is not considering a forcastin approach (that would be interesting to study further but it’s out of scope the present project).

dataset <- dataset[, p.validelements > 0.05]  
testset <- testset[, p.validelements > 0.05]  
  
## trimming the first 7 elements  
  
dataset <- dataset[-c(1:7)]  
testset <- testset[-c(1:7)]

The final data.frame now have the following sizes:

## [1] 19622 53

## [1] 20 53

As we can see, the dataset dimension, and so its complexity, has been reduced make it also more parsimonious in its analisys.

### Creating Test and Validation Dataset

We split the dataset in two, of which will be used to train the different models and for validating them.

set.seed(123)  
  
inTrain <- createDataPartition(dataset$classe, p = 0.7, list = FALSE)  
train.set <- dataset[inTrain,]  
validation.set <- dataset[-inTrain,]  
  
dim(train.set)

## [1] 13737 53

dim(validation.set)

## [1] 5885 53

The target variable for our analysis is the feature classe that shows the following distribution.

Percentage of classes frequencies in the different datasets

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Freq Dataset | % | Freq Training set | % | Freq Testing set | % |
| A | 5580 | 28.4 | 3906 | 28.4 | 1674 | 28.4 |
| B | 3797 | 19.4 | 2658 | 19.3 | 1139 | 19.4 |
| C | 3422 | 17.4 | 2396 | 17.4 | 1026 | 17.4 |
| D | 3216 | 16.4 | 2252 | 16.4 | 964 | 16.4 |
| E | 3607 | 18.4 | 2525 | 18.4 | 1082 | 18.4 |

The class distribution is almost balanced except for the Class A that contains a slight higher number of samples. As we can see the Class distribuition has been preserved by the createDataPartition function.

# Model Cross Validation

As we will compare different algorithms, a preset Cross Validation parameter is set for all different models. A basic cross validation choise for this kind of dataset is 5-fold cross-validation to estimate accuracy. In order to seek a better estimate, each algorithm will be repeated 3 times.

control <- trainControl(method = "repeatedcv",   
 number = 5,   
 repeats = 3,  
 verboseIter = TRUE)  
metric <- "Accuracy"

## Classification Tree

The first model is the simplest one: a classification tree. I am using the train function from caret library using rpart method.

set.seed(111)  
  
start.time <- Sys.time()  
mod.CT <- train(classe ~.,   
 data = train.set,  
 method = 'rpart',  
 tuneLength = 25,  
 trControl = control,  
 metric = metric)  
  
end.time <- Sys.time()  
time.takenCT <- end.time - start.time  
time.takenCT  
  
save(mod.CT, file = "modeCT.RData")  
save(time.takenCT, file = "timeCT.RData")

With an accuracy measured on the validation set

confusionMatrix(predict(mod.CT,newdata = validation.set),  
 validation.set$classe)$overall

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Accuracy | Kappa | AccuracyLower | AccuracyUpper | AccuracyNull | AccuracyPValue | McnemarPValue |
| 0.806 | 0.755 | 0.796 | 0.816 | 0.284 | 0 | 0 |

## Random Forest

set.seed(111)  
  
start.time <- Sys.time()  
mod.RF <- train(classe ~.,  
 data = train.set,  
 method = 'rf',  
 trControl = control,  
 metric = metric,  
 tuneLength = 25)  
  
end.time <- Sys.time()  
timeRF.taken <- end.time - start.time  
timeRF.taken  
  
save(mod.RF, file = "modRF.RData")  
save(timeRF.taken, file = "timeRF.RData")

Gives the following accuracy:

confusionMatrix(predict(mod.RF, validation.set),   
 validation.set$classe)$overall

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Accuracy | Kappa | AccuracyLower | AccuracyUpper | AccuracyNull | AccuracyPValue | McnemarPValue |
| 0.995 | 0.993 | 0.993 | 0.996 | 0.284 | 0 | NaN |

## Boosting

set.seed(111)  
  
start.time <- Sys.time()  
mod.Boosting <- train(classe ~.,  
 data = train.set,  
 method = "gbm",  
 trControl = control,  
 metric = metric,  
 verbose = FALSE)  
  
end.time <- Sys.time()  
time.Boosting <- end.time - start.time  
time.Boosting

With accuracy:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Accuracy | Kappa | AccuracyLower | AccuracyUpper | AccuracyNull | AccuracyPValue | McnemarPValue |
| 0.964 | 0.955 | 0.959 | 0.969 | 0.284 | 0 | 0 |

### Bagging

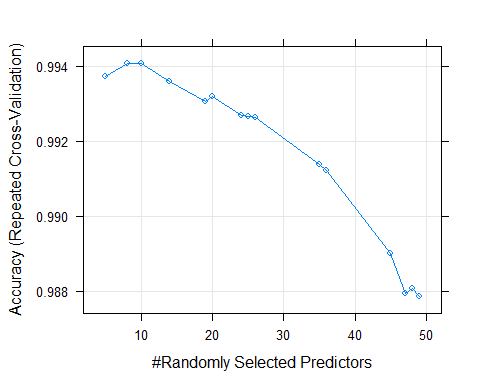
set.seed(111)  
  
start.time <- Sys.time()  
mod.Bagging <- train(classe ~.,  
 data = train.set,  
 method = "treebag",  
 trControl = control,  
 metric = metric)  
  
end.time <- Sys.time()  
time.Bagging <- end.time - start.time

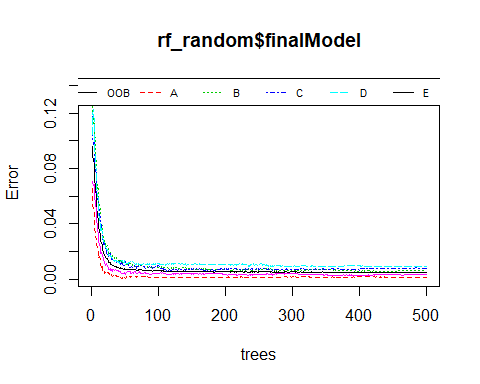
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Accuracy | Kappa | AccuracyLower | AccuracyUpper | AccuracyNull | AccuracyPValue | McnemarPValue |
| 0.984 | 0.98 | 0.981 | 0.987 | 0.284 | 0 | 0.001 |

# Choosing the model

As we can see from the accurancy measured on the testing set, the Random Forest approach looks preferible with respect the other ones. The accuracy is already quite high (about ). The next model tries to tune further the Random Forest model by estimating a more approrpiate value for ntree and mtry parameters using a random search. We will use the model to estimate the parameters’ tree as well.

controlRS <- trainControl(method="repeatedcv",   
 number=10,   
 repeats=3,   
 search="random",  
 verboseIter = TRUE)  
set.seed(111)  
  
rf\_random <- train(classe~.,   
 data=train.set,   
 method="rf",   
 metric=metric,   
 tuneLength=20,   
 trControl=controlRS)  
  
save(rf\_random, file = "rf\_random.RData")





The latest 2 plots suggests that a ntree = 100 and mtry = 10 should speed up the processing while assuring a better accuracy. The following is the model with these parameters and its accuracy on the testing set.

set.seed(111)  
mod.RFfinal <- train(classe ~., data = train.set,  
 method = "rf",  
 ntree = 100,  
 tuneGrid = data.frame(mtry=10),  
 trControl = trainControl(method = "repeatedcv",   
 number=10,  
 repeats=3,  
 verboseIter = TRUE))  
  
confusionMatrix(predict(mod.RFfinal, validation.set),   
 validation.set$classe)$overall

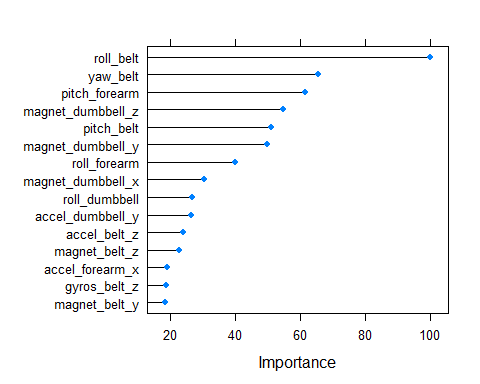
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Accuracy | Kappa | AccuracyLower | AccuracyUpper | AccuracyNull | AccuracyPValue | McnemarPValue |
| 0.995 | 0.994 | 0.993 | 0.997 | 0.284 | 0 | NaN |

And further by class.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Sensitivity | Specificity | Pos Pred Value | Neg Pred Value | Precision | Recall | F1 | Prevalence | Detection Rate | Detection Prevalence | Balanced Accuracy |
| Class: A | 1.000 | 0.999 | 0.998 | 1.000 | 0.998 | 1.000 | 0.999 | 0.284 | 0.284 | 0.285 | 1.000 |
| Class: B | 0.993 | 0.999 | 0.997 | 0.998 | 0.997 | 0.993 | 0.995 | 0.194 | 0.192 | 0.193 | 0.996 |
| Class: C | 0.997 | 0.996 | 0.982 | 0.999 | 0.982 | 0.997 | 0.989 | 0.174 | 0.174 | 0.177 | 0.997 |
| Class: D | 0.990 | 0.999 | 0.996 | 0.998 | 0.996 | 0.990 | 0.993 | 0.164 | 0.162 | 0.163 | 0.994 |
| Class: E | 0.993 | 1.000 | 1.000 | 0.998 | 1.000 | 0.993 | 0.996 | 0.184 | 0.182 | 0.182 | 0.996 |

As we can see from the table, the improvement is really minimal in terms of accuracy although the algorithm speed is improved a lot.

The following plot shows also the top 10 features in terms of importance.



# Applying the model to the test set

Finally, the model is applied to predict the data in the test set.

predict(mod.RF, newdata = testset)

## [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

#knitr::kable(t(predict(mod.RF, newdata = testset)))

# References

* The data used are part of the [WLE dataset](http://groupware.les.inf.puc-rio.br/har)
* Random Forest tuning has been inspired by [James Brownlee blog] (<https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/>)
* A very helpful resource is [Applied Predictive Modeling](http://appliedpredictivemodeling.com/) (both book and related blog)