Practical Machine Learning - Final Assignment

Paolo Coraggio

27/12/2019

Project Outline

For the Coursera Practical Machine Learning final assignment I designed and implemented a model using data from accelerometers placed on the belt, forearm, arm, and dumbell of 6 participants of an experiment to predict the manner in which they did the exercise.

Using the data collected in http://groupware.les.inf.puc-rio.br/har, 4 different predictive models (namely using Regression Tree, Random Forest, Bagging and Boosting algorithm) have been built and compared using a similar setting. Then, the model that resulted with higerh accuracy was further improved and, finally, used on the test dataset.

The workflow of this project is basically:

- loading and filtering the data
- creating a testing and validating test
- building four different models and calculating their accuracy against the validation set
- choosing the final model
- using the final model on the provodided test set

Exploratory data analysis

This section describes how the data has been loaded into R data.frame structures and, from the raw database, the process for extracting the features to build the predictive models.

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 test sets.

The datasets have the following size:

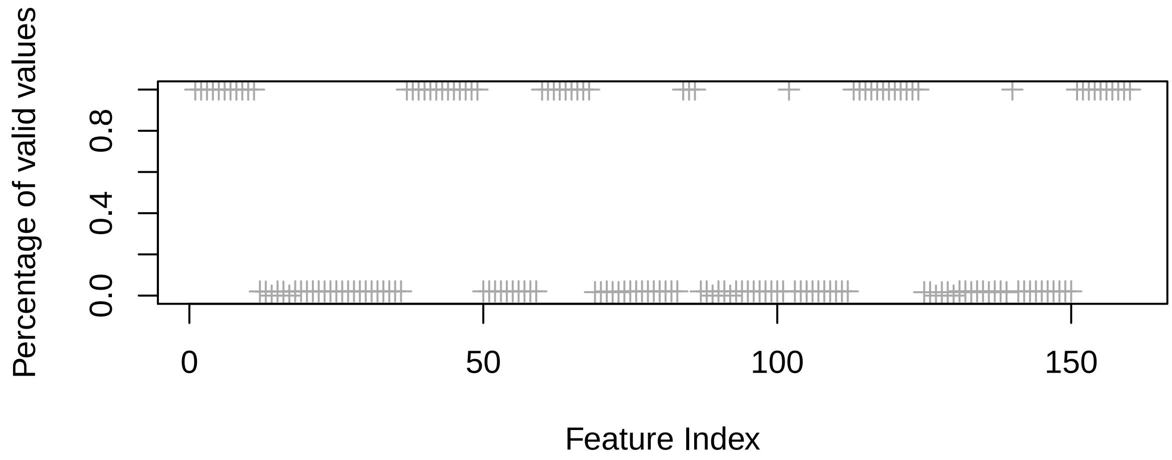
```
## [1] 19622   160
```

```
## [1]  20 160
```

The `dataset` data frame consists of 19622 datapoint of 160 recorded features and the `testset` it's just 20 data points. `testset` contains 20 unknown test cases to be predicted by the model.

Feature Extraction

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



The plot shows that data.frame variables (features) contain or 100% valide data or very few (less than 5% of valide data). The features containing less the 5% of data will be discharged.

Moreover, the first 7 features contain temporal information that has been chosen not to be considered in this project (a forcasting approach would be more suitable).

```
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 making also more parsimonious its analisys.

Test and Validation Dataset

The `dataset` has been splitted in two, 70% to train the different models and 30% 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
```

```r
dim(validation.set)
```

```
## [1] 5885   53
```

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

Table 1: 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.

Predictive models for the dataset

This section is about how 4 different predictive models have been designed and implemented in order to chose the most promising one

Cross Validation

As we will compare different algorithms, a preset Cross Validation parameter is set for all different models. Since the training dataset contains a sufficient number of points, 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 on each folder.

```r
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.

```r
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:

```
round(confusionMatrix(predict(mod.Boosting, validation.set),
                validation.set$classe)$overall,3)
```

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

round(confusionMatrix(predict(mod.Bagging, validation.set),
            validation.set$classe)$overall,3)
```

With an accuracy:

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

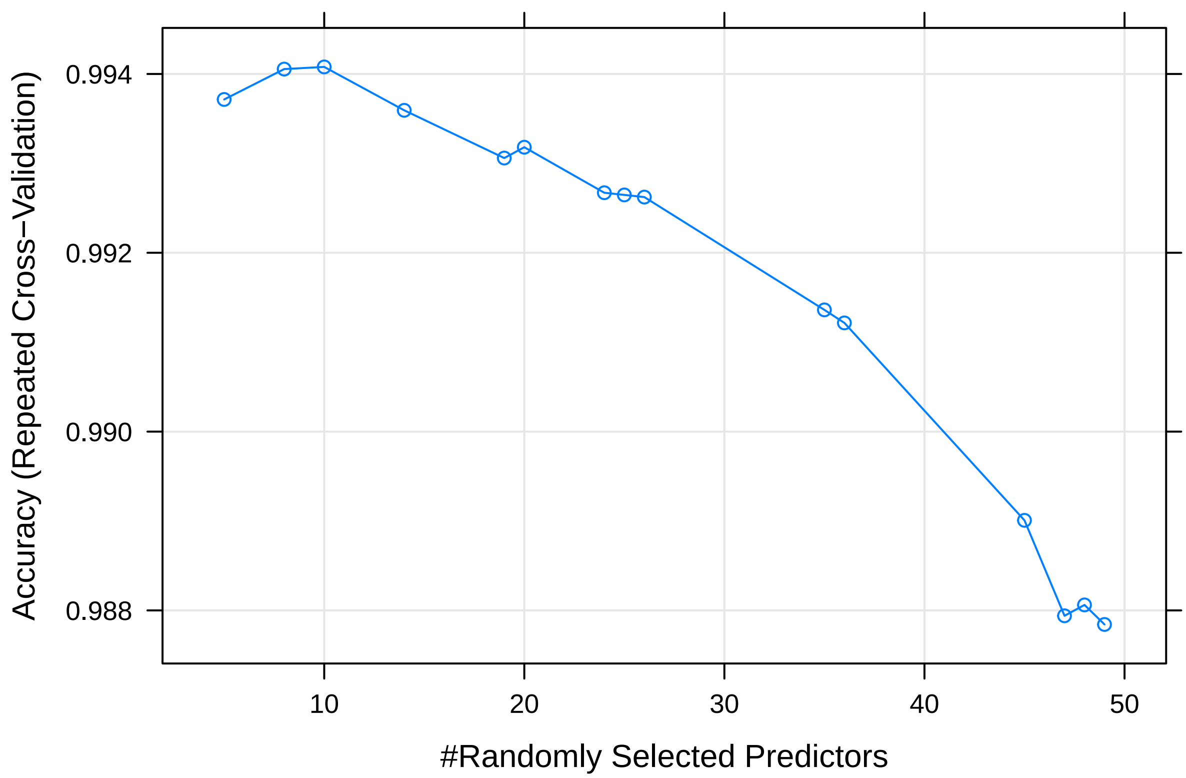
Choosing the model

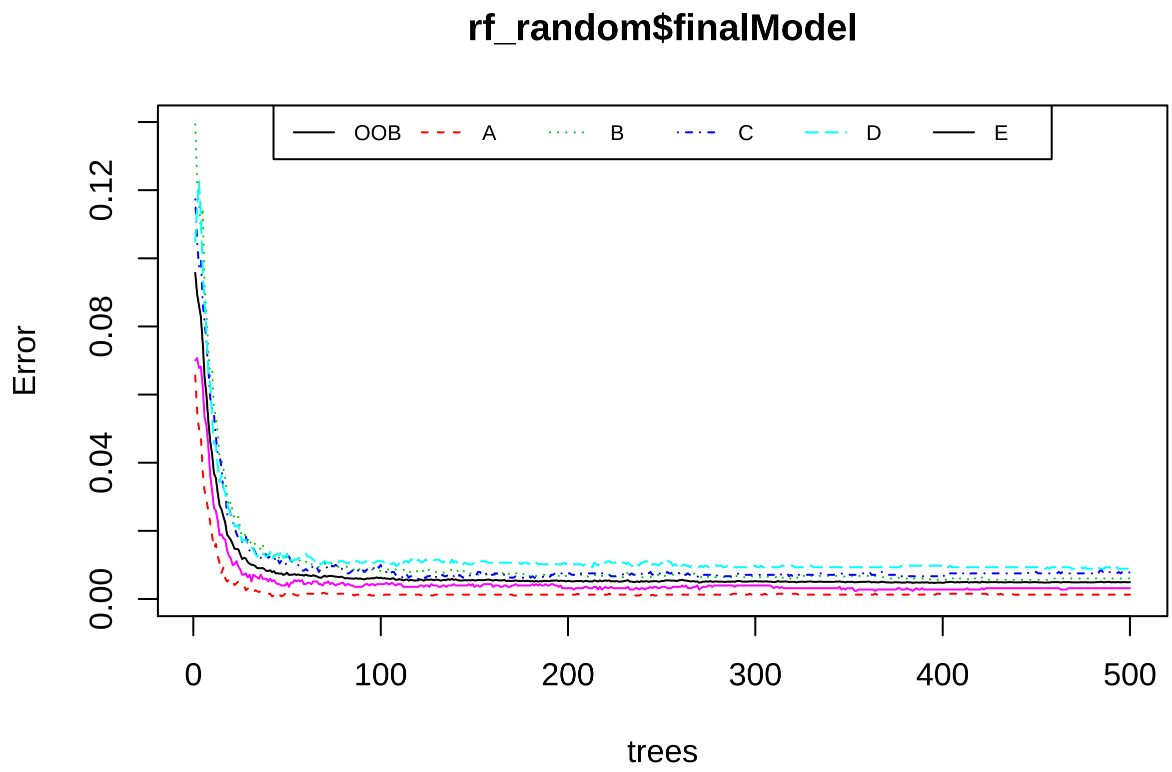
As we can see from the accurancy measured on the testing set, the Random Forest approach looks preferible with respect the other ones. The accuracy is already quite high (about 99.450%). The next model tries to tune further the Random Forest model by estimating a more approrpiate value for `ntree` and `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

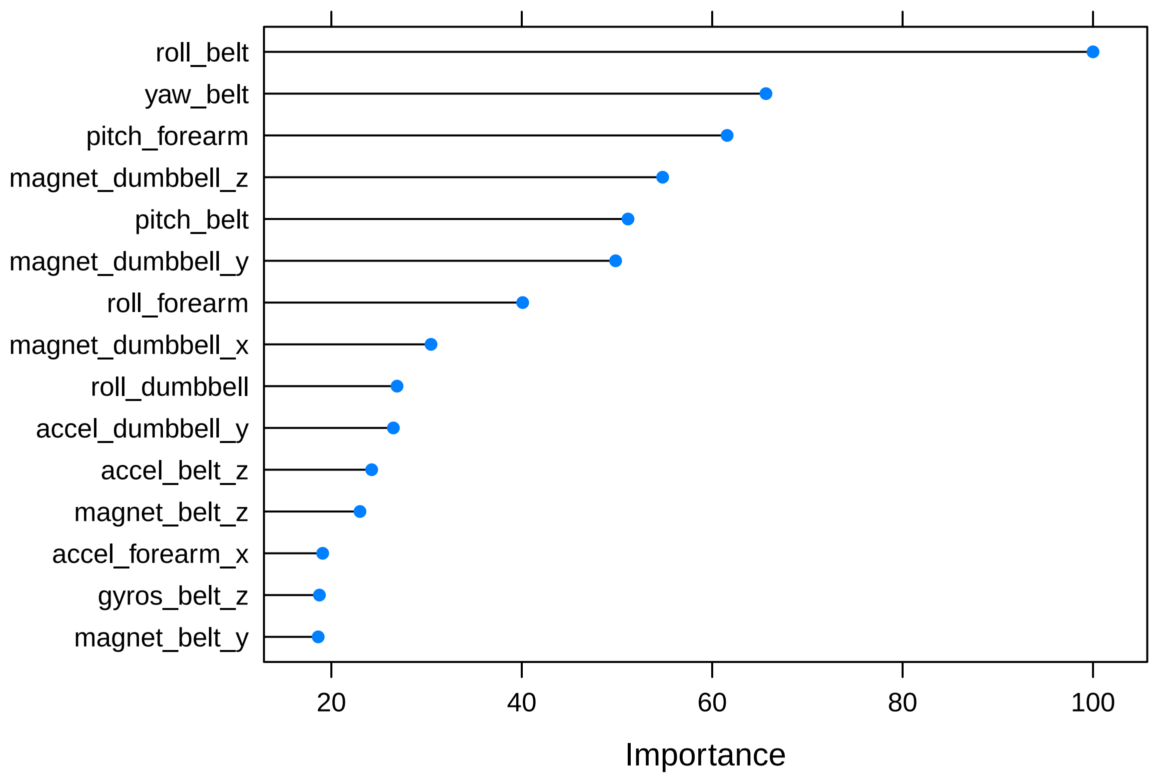
And further by class.

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detectio
Class: A	1.000	0.999	0.998	1.000	0.998	1.000	0.999	0.284	
Class: B	0.993	0.999	0.997	0.998	0.997	0.993	0.995	0.194	

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detectio
Class: C	0.997	0.996	0.982	0.999	0.982	0.997	0.989	0.174	
Class: D	0.990	0.999	0.996	0.998	0.996	0.990	0.993	0.164	
Class: E	0.993	1.000	1.000	0.998	1.000	0.993	0.996	0.184	

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 15 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
```

References and Future Work

- The data used are part of the WLE dataset

- For the Random Forest tuning with caret I read [James Brownlee blog page] (https://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/)
- A very helpful resource for this project has been Applied Predictive Modeling by Max Kuhn and Kjell Johnson (both book and related blog)

I will keep the project updated on the GitHub repository as I will use the proposed assignment to perform further analysis (e.g. PCA and prediction based on temporal series).