ML-course project

A.Izquierdo

5/10/2020

Executive Summary

In this data analysis we aim predict the way in which a set of individuals practised barbell lifts. For this purpose we will generate a Machine Learning Model that we'll validate, analyze and justify. Data are based on the Human Activity Recognition group (http://groupware.les.inf.puc-rio.br/har) and can be found in these two links:

- Testing data (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv)
- Training data (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)

The goal of the model is to predict the training "classe" (A,B,C,D or E) based on the set of variables available in teh file. We'll follow the standard procedure base on data exploration, cleansing, model training, testing and validation.

Libraries used

We load into R the packages used

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.6.3
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.6.3
```

```
## Loaded gbm 2.1.5
```

```
library(AppliedPredictiveModeling)
```

```
## Warning: package 'AppliedPredictiveModeling' was built under R version 3.6.3
```

Test and training data sets

We first load the training and test datasets. Notice we've forced N/A strings to be "NA", "" and "#DIV/0!" because in a pre-exploratory analysis we've detected this N/A values. In this previous analysis we've also seen that columns 8-160 contain variables and columns 1-7 contain dimensions.

```
training <- read.csv("C:/Users/agustin.izquierdo/Documents/R/coursera/Practical Machine Learn
ing/pml-training.csv",na.strings=c("NA","","#DIV/0!"))
testing <- read.csv("C:/Users/agustin.izquierdo/Documents/R/coursera/Practical Machine Learni
ng/pml-testing.csv",na.strings=c("NA","","#DIV/0!"))
```

Exploratory analysis

The training data has 19622 observations 160 variables and 5 measured classes.

```
summary(training$classe)
```

```
##    A    B    C    D    E 
## 5580 3797 3422 3216 3607
```

```
dim(training)
```

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

Data cleansing

We'll clean the dataset in three steps. First we get rid of the first **seven** columns, which contain dimensions we'll not use for the prediction (we'll use only the **classe** dimension and the variables). These dimensions contain information such as row number, username, timestamp and 'window'.

```
training <- training[,-(1:7)]
```

We have already mentioned we set to N/A all the empty characters. No we get rid of all those variables which are empty.

```
nonNAcols <- colSums(is.na(training))==0
training <- training[, nonNAcols]
dim(training)
```

```
## [1] 19622    53
```

We move from 160 columns in the original dataset to 53

Finally, we look for the non-zero variance predictors, with the NearZeroVar function. From its outcome, we take from the training dataset only those variables which have non-zero variance

```
NZV<-nearZeroVar(training, saveMetrics = TRUE)
training <- training[ , NZV$nzv==FALSE]
dim(training)
```

```
## [1] 19622    53
```

We keep the same 53 columns, therefore out of these non of them has zero variance.

From this clean dataset we split the dataset into two, to train and validate the model

We apply the same cleansing for the testing dataset, so both are comparable.

```
dim(testing)
```

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

```
testing <- testing[,-(1:7)]
testing <- testing[, nonNAcols]
testing <- testing[ , NZV$nzv==FALSE]
dim(testing)
```

```
## [1] 20 53
```

Now we’re ready to start modelling.

Model generation

We first set the seed for reproducibility matters and get the test and training datasets. We generate a model based on **random forest** resampling 10 times (cross-validations)

```
set.seed(1977)
RFmodelFit<- train(classe ~ ., data=trainingDS, method="rf", trControl = trainControl(method=
"cv"),number=10)
RFmodelFit
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 12364, 12362, 12365, 12362, 12363, 12363, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9916291  0.9894096
##   27    0.9916293  0.9894107
##   52    0.9842767  0.9801080
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.
```

We already see from this output that the model is quite accurate (0.99 for 2 variables randomly sampled as candidates at each split). Once we have the model, let’s see how accurate it is

```
RFmodelPred<-predict(RFmodelFit,training)
confusionMatrix(RFmodelPred,training$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 5579    4    0    0    0
##          B    1 3793    6    0    0
##          C    0    0 3413   11    7
##          D    0    0    3 3204    1
##          E    0    0    0    1 3599
##
## Overall Statistics
##
##                Accuracy : 0.9983
##                  95% CI : (0.9976, 0.9988)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9978
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9998   0.9989   0.9974   0.9963   0.9978
## Specificity            0.9997   0.9996   0.9989   0.9998   0.9999
## Pos Pred Value         0.9993   0.9982   0.9948   0.9988   0.9997
## Neg Pred Value         0.9999   0.9997   0.9994   0.9993   0.9995
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1933   0.1739   0.1633   0.1834
## Detection Prevalence   0.2845   0.1937   0.1749   0.1635   0.1835
## Balanced Accuracy      0.9998   0.9993   0.9981   0.9980   0.9989
```

We see the random forest model fits extremely well the validation dataset (0.9978 accuracy)

Let’s try a **boost model**.

```
BoostmodelFit<- train(classe ~ ., data=trainingDS, method="gbm",verbose=FALSE)
BoostmodelFit
```

```
## Stochastic Gradient Boosting
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 13737, 13737, 13737, 13737, 13737, 13737, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7511386  0.6844881
##   1                  100      0.8173842  0.7689171
##   1                  150      0.8506647  0.8110549
##   2                   50      0.8517065  0.8122008
##   2                  100      0.9047184  0.8794566
##   2                  150      0.9294466  0.9107483
##   3                   50      0.8931720  0.8648201
##   3                  100      0.9389584  0.9227904
##   3                  150      0.9578140  0.9466437
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

We see the accuracy is lower than the random forest model. Let’s see it detail:

```
BoostmodelPred<-predict(BoostmodelFit,validationDS)
confusionMatrix(BoostmodelPred,validationDS$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1650   32    0    1    2
##          B   17 1086   36    3   10
##          C    4   20  973   33   15
##          D    1    1   12  915   11
##          E    2    0    5   12 1044
##
## Overall Statistics
##
##                Accuracy : 0.9631
##                  95% CI : (0.958, 0.9678)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9533
##
##  Mcnemar's Test P-Value : 2.532e-05
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9857   0.9535   0.9483   0.9492   0.9649
## Specificity            0.9917   0.9861   0.9852   0.9949   0.9960
## Pos Pred Value         0.9792   0.9427   0.9311   0.9734   0.9821
## Neg Pred Value         0.9943   0.9888   0.9890   0.9901   0.9921
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2804   0.1845   0.1653   0.1555   0.1774
## Detection Prevalence   0.2863   0.1958   0.1776   0.1597   0.1806
## Balanced Accuracy      0.9887   0.9698   0.9668   0.9720   0.9805
```

So the boosting model has a 0.96 accuracy versus the 0.9985 accuracy of the random forest one. Therefore, the preferred model would be the random forest.

Prediction on test dataset

Let’s take a look at test results of both models and compare them.

```
RFTest <- predict(RFmodelFit, testing)
BoostTtest<- predict(BoostmodelFit, testing)
table(RFTest,BoostTtest)
```

```
##       BoostTtest
## RFTest A B C D E
##      A 7 0 0 0 0
##      B 0 8 0 0 0
##      C 0 0 1 0 0
##      D 0 0 0 1 0
##      E 0 0 0 0 3
```

We see that both models give the same result on the testing dataset (we get figures only in the diagonal of the table). By the way, this is not the case if we make the same exercise on any of the training dataset

```
RFTest2 <- predict(RFmodelFit, trainingDS)
BoostTtest2<- predict(BoostmodelFit, trainingDS)
table(RFTest2,BoostTtest2)
```

```
##        BoostTtest2
## RFTest2    A    B    C    D    E
##       A 3866   29    7    2    2
##       B   68 2549   41    0    0
##       C    0   59 2315   20    2
##       D    1    3   62 2174   12
##       E    2   16   10   25 2472
```

Finally, we write the output of the random forest model on a text file:

```
writeLines(as.character(RFTest),"modelPredictionTest.txt")
```