Machine learning project

Tanguy

April 6, 2017

Executive summary

The goal of this project is to predict the outcome variable called classe which can take value from class A to class E. Class A is a well-performed exercise and the other classes correspond to execution mistakes in doing the exercise. The first step prior to any use of algorithm is to clean the data.The training file is then divided in a training and validation set. Decisions are made taking into account the necessity to decrease the computation time. Then, the two more "famous" model studied in the course, namely random forest and boosting, have their accuracy compared on the validation dataset. The best model is random forest and it is applied to the testing set.

1. Loading, viewing and cleaning data

```r
library(ggplot2)
setwd("~/Cours/Machine learning/Assignment")
## Loading data replacing missing values by NA
trainingdata <- read.csv("pml-training.csv",na.strings=c("NA","#DIV/0!", ""))
testingdata <- read.csv("pml-testing.csv",na.strings=c("NA","#DIV/0!", ""))
## Data overview
dim(trainingdata)
```

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

```r
dim(testingdata)
```

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

```r
##View(trainingdata)
```

The dataset contains 160 columns and a tremendous number of observations and also a lot of NA values in some columns.

```r
set.seed(500)
## Data cleaning
##Let's focus only on data from sensors (no user variation from instance)
trainingdata <-trainingdata[,-c(1:7)]
## Remove columns with not so much variation
suppressMessages(library(caret))
columnsLowVariance <- nearZeroVar(trainingdata, saveMetrics = TRUE)
trainingdata <- trainingdata[, columnsLowVariance$nzv==FALSE]
dim(trainingdata)
```

```
## [1] 19622   118
```

```r
##Remove columns with a lot of NA
treshold <- dim(trainingdata)[1] * 0.6
columns <- !apply(trainingdata, 2, function(x) sum(is.na(x)) > treshold  || sum(x=="") > treshold)
trainingdata <- trainingdata[,columns]
dim(trainingdata) ##This is still too many
```

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

Let's try to run random forest with a small set of data, to evaluate importance of regressors and to remove the least important ones to keep "only" 20 variables.

```
suppressMessages(library(randomForest))
inTrain0 <- createDataPartition(trainingdata$classe, p = 0.1,list=FALSE)
analysis <- trainingdata[inTrain0,]
modFit0 <- train(classe ~ .,data=analysis,method="rf",importance = TRUE,prox=TRUE)
```

```
imp <- varImp(modFit0)
print(imp)
```

```
## rf variable importance
##
##   variables are sorted by maximum importance across the classes
##   only 20 most important variables shown (out of 52)
##
##                          A     B     C     D      E
## roll_belt            80.69 91.82 74.89 81.45 100.00
## magnet_dumbbell_z    86.92 60.89 74.11 62.87  49.10
## pitch_forearm        83.46 52.00 81.47 70.33  56.27
## magnet_dumbbell_y    73.18 67.91 69.04 66.53  54.15
## pitch_belt           27.46 56.53 29.34 33.28  24.92
## yaw_belt             47.61 41.28 46.50 51.39  45.29
## roll_forearm         47.89 35.74 42.70 33.46  33.88
## roll_dumbbell        38.44 37.49 46.82 42.07  36.86
## yaw_dumbbell         17.51 44.96 24.95 21.24  23.40
## accel_dumbbell_y     32.07 28.37 42.36 35.65  35.73
## magnet_belt_z        26.56 40.54 26.05 33.61  30.48
## accel_forearm_x      18.79 25.27 24.51 39.85  25.57
## total_accel_dumbbell 16.87 31.52 15.14 36.16  30.52
## accel_dumbbell_z     24.12 30.35 25.98 24.37  30.98
## gyros_dumbbell_y     30.66 15.60 25.27 17.37  10.79
## magnet_dumbbell_x    23.93 22.25 30.12 26.38  19.06
## magnet_belt_y        21.51 28.55 27.36 27.21  24.24
## yaw_arm              28.00 16.11 25.99 26.30  16.11
## magnet_belt_x        14.64 27.74 24.47 21.22  16.17
## accel_belt_z         19.08 20.26 21.91 26.27  16.95
```

Let's keep the 20 most important columns

```
trainingdata <-trainingdata[,-c(4:9,14,15,17:20,21:23,24:26,28,31,33,34,42,43,44:46,48,49,50:52)]
dim(trainingdata)
```

```
## [1] 19622    21
```

Perparing datasets:

```
inTrain <- createDataPartition(trainingdata$classe, p = 0.7,list=FALSE)
training <- trainingdata[ inTrain,]
validating <- trainingdata[ -inTrain,]
testing <- testingdata[,c(8,9,10,42:45,48,84,86,102,114,117:123,154,160)]
```

2. Use of algorithms

The 2 most performant algorithms seen in the course are random forest and boosting. Let's use them now on the training set, compare their performance on the validation set and choose the best one to be used on the test set.

```r
suppressMessages(library(randomForest))
suppressMessages(library(caret))
library(ggplot2)
suppressMessages(library(gbm))
##calculate models with training set
modFit_rf <- train(classe ~ .,data=training,method="rf")
modFit_boosting <- train(classe ~ .,data=training, method="gbm",verbose=FALSE)
```

```
## Loading required package: plyr
```

```r
##predict with validation set
prediction_rf <- predict(modFit_rf, validating)
prediction_boosting <- predict(modFit_boosting, validating)
##Assessing models
confusionMatrix(prediction_rf, validating$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1669   13    0    0    0
##          B    5 1115   11    1    1
##          C    0   11 1013    9    2
##          D    0    0    2  953    5
##          E    0    0    0    1 1074
##
## Overall Statistics
##
##                Accuracy : 0.9896
##                  95% CI : (0.9867, 0.9921)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9869
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9970   0.9789   0.9873   0.9886   0.9926
## Specificity            0.9969   0.9962   0.9955   0.9986   0.9998
## Pos Pred Value         0.9923   0.9841   0.9787   0.9927   0.9991
## Neg Pred Value         0.9988   0.9949   0.9973   0.9978   0.9983
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2836   0.1895   0.1721   0.1619   0.1825
## Detection Prevalence   0.2858   0.1925   0.1759   0.1631   0.1827
## Balanced Accuracy      0.9970   0.9876   0.9914   0.9936   0.9962
```

```r
confusionMatrix(prediction_boosting, validating$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1629   44    0    0    1
##          B   28 1040   30    3   15
##          C    3   46  978   17    8
##          D    7    6   18  935   17
##          E    7    3    0    9 1041
##
## Overall Statistics
##
##                Accuracy : 0.9555
##                  95% CI : (0.9499, 0.9606)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9437
##  Mcnemar's Test P-Value : 1.169e-05
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9731   0.9131   0.9532   0.9699   0.9621
## Specificity            0.9893   0.9840   0.9848   0.9902   0.9960
## Pos Pred Value         0.9731   0.9319   0.9297   0.9512   0.9821
## Neg Pred Value         0.9893   0.9792   0.9901   0.9941   0.9915
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2768   0.1767   0.1662   0.1589   0.1769
## Detection Prevalence   0.2845   0.1896   0.1788   0.1670   0.1801
## Balanced Accuracy      0.9812   0.9485   0.9690   0.9801   0.9791
```

To conclude, the best model is random forest with accuracy equal to 0.99. Boosting has also a quite high accuracy equal to 0.96. We apply the random forest model to the testing datatset.

```r
my_prediction <- predict(modFit_rf, testing)
my_prediction
```

```
##  [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
```