

Practical ML

José Antonio García Ramírez

January 19, 2017

Get the Data

Since the Train and Test sets are defined in this exercise, we will not say anything about the bias that can contain the sample that forms the Train set but even though in an ideal case, with enough computer equipment, we would like to use the one-leave-out method to train the models, since the final part of the exercise consists of predicting 20 observations, due to run-time issues. It was decided to use only the 10-fold cross validation, which is faster than one-leave-out validation and also presents less risk in overtraining the models with the Trainset.

```
Train <- read.csv('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv')
Test <- read.csv('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv')
```

Preprocess and pruning

Then we get the datasets, directly from the web addresses, and then we notice that many of the variables containing both datasets present a considerable amount of NA and we remove those variables with little information

```
library(ggplot2)
NApercentage <- function(column)
{
  return(mean(is.na(column)))
}
listNA <- apply(Train, 2, NApercentage)
listNA
```

```
##              X              user_name      raw_timestamp_part_1
##      0.0000000      0.0000000      0.0000000
## raw_timestamp_part_2      cvtd_timestamp      new_window
##      0.0000000      0.0000000      0.0000000
##      num_window      roll_belt      pitch_belt
##      0.0000000      0.0000000      0.0000000
##      yaw_belt      total_accel_belt      kurtosis_roll_belt
##      0.0000000      0.0000000      0.0000000
##      kurtosis_pitch_belt      kurtosis_yaw_belt      skewness_roll_belt
##      0.0000000      0.0000000      0.0000000
##      skewness_roll_belt.1      skewness_yaw_belt      max_roll_belt
##      0.0000000      0.0000000      0.9793089
##      max_pitch_belt      max_yaw_belt      min_roll_belt
##      0.9793089      0.0000000      0.9793089
##      min_pitch_belt      min_yaw_belt      amplitude_roll_belt
##      0.9793089      0.0000000      0.9793089
##      amplitude_pitch_belt      amplitude_yaw_belt      var_total_accel_belt
##      0.9793089      0.0000000      0.9793089
##      avg_roll_belt      stddev_roll_belt      var_roll_belt
```

##	0.9793089	0.9793089	0.9793089
##	avg_pitch_belt	stddev_pitch_belt	var_pitch_belt
##	0.9793089	0.9793089	0.9793089
##	avg_yaw_belt	stddev_yaw_belt	var_yaw_belt
##	0.9793089	0.9793089	0.9793089
##	gyros_belt_x	gyros_belt_y	gyros_belt_z
##	0.0000000	0.0000000	0.0000000
##	accel_belt_x	accel_belt_y	accel_belt_z
##	0.0000000	0.0000000	0.0000000
##	magnet_belt_x	magnet_belt_y	magnet_belt_z
##	0.0000000	0.0000000	0.0000000
##	roll_arm	pitch_arm	yaw_arm
##	0.0000000	0.0000000	0.0000000
##	total_accel_arm	var_accel_arm	avg_roll_arm
##	0.0000000	0.9793089	0.9793089
##	stddev_roll_arm	var_roll_arm	avg_pitch_arm
##	0.9793089	0.9793089	0.9793089
##	stddev_pitch_arm	var_pitch_arm	avg_yaw_arm
##	0.9793089	0.9793089	0.9793089
##	stddev_yaw_arm	var_yaw_arm	gyros_arm_x
##	0.9793089	0.9793089	0.0000000
##	gyros_arm_y	gyros_arm_z	accel_arm_x
##	0.0000000	0.0000000	0.0000000
##	accel_arm_y	accel_arm_z	magnet_arm_x
##	0.0000000	0.0000000	0.0000000
##	magnet_arm_y	magnet_arm_z	kurtosis_roll_arm
##	0.0000000	0.0000000	0.0000000
##	kurtosis_pitch_arm	kurtosis_yaw_arm	skewness_roll_arm
##	0.0000000	0.0000000	0.0000000
##	skewness_pitch_arm	skewness_yaw_arm	max_roll_arm
##	0.0000000	0.0000000	0.9793089
##	max_pitch_arm	max_yaw_arm	min_roll_arm
##	0.9793089	0.9793089	0.9793089
##	min_pitch_arm	min_yaw_arm	amplitude_roll_arm
##	0.9793089	0.9793089	0.9793089
##	amplitude_pitch_arm	amplitude_yaw_arm	roll_dumbbell
##	0.9793089	0.9793089	0.0000000
##	pitch_dumbbell	yaw_dumbbell	kurtosis_roll_dumbbell
##	0.0000000	0.0000000	0.0000000
##	kurtosis_pitch_dumbbell	kurtosis_yaw_dumbbell	skewness_roll_dumbbell
##	0.0000000	0.0000000	0.0000000
##	skewness_pitch_dumbbell	skewness_yaw_dumbbell	max_roll_dumbbell
##	0.0000000	0.0000000	0.9793089
##	max_pitch_dumbbell	max_yaw_dumbbell	min_roll_dumbbell
##	0.9793089	0.0000000	0.9793089
##	min_pitch_dumbbell	min_yaw_dumbbell	amplitude_roll_dumbbell
##	0.9793089	0.0000000	0.9793089
##	amplitude_pitch_dumbbell	amplitude_yaw_dumbbell	total_accel_dumbbell
##	0.9793089	0.0000000	0.0000000
##	var_accel_dumbbell	avg_roll_dumbbell	stddev_roll_dumbbell
##	0.9793089	0.9793089	0.9793089
##	var_roll_dumbbell	avg_pitch_dumbbell	stddev_pitch_dumbbell
##	0.9793089	0.9793089	0.9793089
##	var_pitch_dumbbell	avg_yaw_dumbbell	stddev_yaw_dumbbell

```
##          0.9793089          0.9793089          0.9793089
##      var_yaw_dumbbell      gyros_dumbbell_x      gyros_dumbbell_y
##          0.9793089          0.0000000          0.0000000
##      gyros_dumbbell_z      accel_dumbbell_x      accel_dumbbell_y
##          0.0000000          0.0000000          0.0000000
##      accel_dumbbell_z      magnet_dumbbell_x      magnet_dumbbell_y
##          0.0000000          0.0000000          0.0000000
##      magnet_dumbbell_z      roll_forearm          pitch_forearm
##          0.0000000          0.0000000          0.0000000
##          yaw_forearm      kurtosis_roll_forearm      kurtosis_pitch_forearm
##          0.0000000          0.0000000          0.0000000
##      kurtosis_yaw_forearm      skewness_roll_forearm      skewness_pitch_forearm
##          0.0000000          0.0000000          0.0000000
##      skewness_yaw_forearm      max_roll_forearm      max_pitch_forearm
##          0.0000000          0.9793089          0.9793089
##      max_yaw_forearm      min_roll_forearm      min_pitch_forearm
##          0.0000000          0.9793089          0.9793089
##      min_yaw_forearm      amplitude_roll_forearm      amplitude_pitch_forearm
##          0.0000000          0.9793089          0.9793089
##      amplitude_yaw_forearm      total_accel_forearm      var_accel_forearm
##          0.0000000          0.0000000          0.9793089
##      avg_roll_forearm      stddev_roll_forearm      var_roll_forearm
##          0.9793089          0.9793089          0.9793089
##      avg_pitch_forearm      stddev_pitch_forearm      var_pitch_forearm
##          0.9793089          0.9793089          0.9793089
##      avg_yaw_forearm      stddev_yaw_forearm      var_yaw_forearm
##          0.9793089          0.9793089          0.9793089
##      gyros_forearm_x      gyros_forearm_y      gyros_forearm_z
##          0.0000000          0.0000000          0.0000000
##      accel_forearm_x      accel_forearm_y      accel_forearm_z
##          0.0000000          0.0000000          0.0000000
##      magnet_forearm_x      magnet_forearm_y      magnet_forearm_z
##          0.0000000          0.0000000          0.0000000
##          classe
##          0.0000000
```

It is decided to eliminate those variables where more than 50% of their contents are NA, and after a manual revision (with the summary function), the “read.csv” function identifies the character “# DIV / 0!” In some columns what originated that those that contain it are read as factors instead of being numerical, all of them also are little informative because they contain a large percentage of NA, also these variables were discarded including variables with temporal information.

```
removeVar <- listNA[listNA > .5]
Vars <- setdiff(colnames(Train), names(removeVar))
#summary(Train[,Vars])
```

```
removeDIV <- c("X", "raw_timestamp_part_1", "raw_timestamp_part_2",
  "cvt_d_timestamp", "kurtosis_roll_belt", "kurtosis_pitch_belt",
  "kurtosis_yaw_belt", "skewness_roll_belt", "skewness_roll_belt.1",
  "skewness_yaw_belt", "max_yaw_belt", "min_yaw_belt",
  "amplitude_yaw_belt", "kurtosis_roll_arm", "kurtosis_pitch_arm",
  "kurtosis_yaw_arm", "skewness_roll_arm", "skewness_pitch_arm",
  "skewness_yaw_arm", "kurtosis_roll_dumbbell",
```

```

      "kurtosis_picth_dumbbell", "kurtosis_yaw_dumbbell",
      "skewness_roll_dumbbell", "skewness_pitch_dumbbell",
      "skewness_yaw_dumbbell", "max_yaw_dumbbell",
      "min_yaw_dumbbell", "amplitude_yaw_dumbbell",
      "kurtosis_roll_forearm", "kurtosis_picth_forearm",
      "kurtosis_yaw_forearm", "skewness_roll_forearm",
      "skewness_pitch_forearm", "skewness_yaw_forearm",
      "max_yaw_forearm", "min_yaw_forearm", "amplitude_yaw_forearm")
Vars <- setdiff(Vars, removeDIV)

```

At this point in the Train dataset set there are no NA values so no pruning is required.

```
sum(is.na(Train[,Vars]))
```

```
## [1] 0
```

It is important to note the distribution of the classes in our dataset, as you can see in table 1, class ‘A’ has approximately 28% of the cases, this induces a lower bound on the performance of the classification algorithms that we will try.

```

library(pander)
a <- table(Train$classe)/ dim(Train)[1]
pander(round(a, 2))

```

A	B	C	D	E
0.28	0.19	0.17	0.16	0.18

Table 1: *classe*’s distribution, target variable, in the training dataset.

Models selection

Given that 55 features are available, some of which are numeric and other factors, to predict one of type factor, variable *classe*, we are faced with a classification problem rather than a regression problem.

Given that the content of most of the available variables is uncertain, three algorithms will be used that are not sensitive to transformations of the variables(including centering and scaling):

- Quadratic discriminant analysis (QDA)
- Random forest
- Support vector machine (SVM)

So, we train the models, first the QDA. A LDA model was also trained but its accuracy is approximately 0.77 and for it was discarded.

```

library(caret)
controlTrain <- trainControl(method = "cv", number = 10, allowParallel = TRUE) #indicating 10-fold
QDA <- train(classe~. , data = Train[, Vars], method = "qda",
             trControl = controlTrain)
QDAClasses <- predict(QDA, newdata = Train[, Vars])
confusionMatrix(QDAClasses, Train$classe)

```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##           A 5305  159    2    3    0
##           B  171 3322  124   17   88
##           C   55  267 3272  340  129
##           D   44   31   17 2831   89
##           E    5   18    7  25 3301
##
## Overall Statistics
##
##           Accuracy : 0.9189
##           95% CI : (0.915, 0.9227)
##           No Information Rate : 0.2844
##           P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.8975
##           McNemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##           Class: A Class: B Class: C Class: D Class: E
## Sensitivity      0.9507  0.8749  0.9562  0.8803  0.9152
## Specificity      0.9883  0.9747  0.9512  0.9890  0.9966
## Pos Pred Value   0.9700  0.8925  0.8053  0.9399  0.9836
## Neg Pred Value   0.9806  0.9701  0.9904  0.9768  0.9812
## Prevalence       0.2844  0.1935  0.1744  0.1639  0.1838
## Detection Rate   0.2704  0.1693  0.1668  0.1443  0.1682
## Detection Prevalence 0.2787  0.1897  0.2071  0.1535  0.1710
## Balanced Accuracy 0.9695  0.9248  0.9537  0.9346  0.9559
```

Surprisingly, the SVM algorithm has less accuracy than the QDA

```
SV <- train(classe ~ ., data = Train[, Vars], method = "svmLinear",
            trControl = controlTrain )
```

```
## Loading required package: kernlab
```

```
##
```

```
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
```

```
##
```

```
##      alpha
```

```
SVClasses <- predict(SV, newdata = Train[, Vars])
confusionMatrix(SVClasses, Train$classe)
```

```
## Confusion Matrix and Statistics
```

```
##
```

```
##           Reference
```

```
## Prediction      A      B      C      D      E
##           A 5207  479  241  158  124
##           B  119 2826  273  111  316
##           C  102  206 2776  353  194
##           D  145   51   73 2500  155
##           E    7  235   59   94 2818
##
## Overall Statistics
##
##           Accuracy : 0.8219
##           95% CI : (0.8165, 0.8272)
##           No Information Rate : 0.2844
##           P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.7738
##           McNemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##           Class: A Class: B Class: C Class: D Class: E
## Sensitivity      0.9332  0.7443  0.8112  0.7774  0.7813
## Specificity      0.9286  0.9482  0.9472  0.9742  0.9753
## Pos Pred Value   0.8386  0.7753  0.7645  0.8550  0.8771
## Neg Pred Value   0.9722  0.9392  0.9596  0.9571  0.9519
## Prevalence       0.2844  0.1935  0.1744  0.1639  0.1838
## Detection Rate   0.2654  0.1440  0.1415  0.1274  0.1436
## Detection Prevalence 0.3164  0.1858  0.1850  0.1490  0.1637
## Balanced Accuracy 0.9309  0.8463  0.8792  0.8758  0.8783
```

And even more surprisingly Random Forest classifies phenomenally.

```
RF <- train(classe ~ ., data = Train[, Vars], method = "rf",
            trControl = controlTrain)
```

```
## Loading required package: randomForest
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
```

```
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
```

```
##
```

```
##      margin
```

```
RFClasses <- predict(RF, newdata = Train[, Vars])
confusionMatrix(RFClasses, Train$classe)
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction    A    B    C    D    E
##           A 5580    0    0    0    0
##           B    0 3797    0    0    0
##           C    0    0 3422    0    0
##           D    0    0    0 3216    0
##           E    0    0    0    0 3607
##
## Overall Statistics
##
##           Accuracy : 1
##           95% CI : (0.9998, 1)
##           No Information Rate : 0.2844
##           P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 1
##           McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##           Class: A Class: B Class: C Class: D Class: E
## Sensitivity           1.0000    1.0000    1.0000    1.0000    1.0000
## Specificity           1.0000    1.0000    1.0000    1.0000    1.0000
## Pos Pred Value        1.0000    1.0000    1.0000    1.0000    1.0000
## Neg Pred Value        1.0000    1.0000    1.0000    1.0000    1.0000
## Prevalence            0.2844    0.1935    0.1744    0.1639    0.1838
## Detection Rate        0.2844    0.1935    0.1744    0.1639    0.1838
## Detection Prevalence  0.2844    0.1935    0.1744    0.1639    0.1838
## Balanced Accuracy     1.0000    1.0000    1.0000    1.0000    1.0000
```

In view of the above, the fabulous performance of RF, it is decided to use the predictions of the SVM and QDA models and then give them as inputs to RandomForest, not to overtrain. The assembled model trained on the Train data set is as follows:

```
dataTrain <- data.frame(SVM = SVClasses, QDA = QDAClasses, classe = Train$classe )
RF.finalTrain <- train(classe ~ ., data = dataTrain, method = "rf")
RF.final <- predict(RF.finalTrain, newdata = dataTrain)
confusionMatrix(RF.final, Train$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##           A 5305  159    2    3    0
##           B  171 3322  124   17   88
##           C   55  267 3272  340  129
##           D   44   31   17 2831   89
##           E    5   18    7  25 3301
##
## Overall Statistics
##
##           Accuracy : 0.9189
```

```
##              95% CI : (0.915, 0.9227)
##      No Information Rate : 0.2844
##      P-Value [Acc > NIR] : < 2.2e-16
##
##              Kappa : 0.8975
##      McNemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##              Class: A Class: B Class: C Class: D Class: E
## Sensitivity      0.9507  0.8749  0.9562  0.8803  0.9152
## Specificity      0.9883  0.9747  0.9512  0.9890  0.9966
## Pos Pred Value   0.9700  0.8925  0.8053  0.9399  0.9836
## Neg Pred Value   0.9806  0.9701  0.9904  0.9768  0.9812
## Prevalence       0.2844  0.1935  0.1744  0.1639  0.1838
## Detection Rate   0.2704  0.1693  0.1668  0.1443  0.1682
## Detection Prevalence 0.2787  0.1897  0.2071  0.1535  0.1710
## Balanced Accuracy 0.9695  0.9248  0.9537  0.9346  0.9559
```

So the predictions for the 20 different test cases are the following:

```
library(knitr)
test.SV <- predict(SV, Test)
test.QDA <- predict(QDA, Test)
test.final <- data.frame(SVM = test.SV, QDA = test.QDA,
                        id = Test$problem_id)
test.final$Output <- predict(RF.finalTrain, test.final)
kable(test.final[,c("id", "Output")])
```

id	Output
1	A
2	A
3	B
4	A
5	A
6	E
7	D
8	B
9	A
10	A
11	B
12	C
13	B
14	A
15	E
16	E
17	A
18	B
19	B
20	B