Practical Machine Learning - Final Assignment

Barbara Froner

3 December 2016

Objective

The objective of this project is to predict the way in which the candidates involved in the Human Activity Recognition research study performed the weight lifting exercises assigned to them. The variable “classe” in the provided training dataset represents the manner in which the exercises were executed and could be predicted in the testing dataset by using any other available variable as predictor.

The Data at Glance

The data provided for the exercise was downloaded from the course web page on the Coursera website https://www.coursera.org (https://www.coursera.org) and read into the R session via *read.csv()*.

```
training <- read.csv( paste( folder, "pml-training.csv", sep = "" ) )
testing <- read.csv( paste( folder, "pml-testing.csv", sep = "" ) )
```

The data provided comprised:

- a *training* dataset of 19622 rows (observations) by 160 columns (variables), used to create the machine learning models;
- a *testing* dataset of 20 rows by 159 columns, where the “classe” variable was missing and had to be predicted using the models.

Given the high number of variables in the dataset it was hard to understand how they all related to each other and to the dependent variable “classe” by simply plotting them with, for example, with cross plots.

By looking at the structure and the summary of the dataset with the commands *str* and *summary* it was possible to see that there were variables of different type, including integer, factor and numeric variables. It was also possible to see that a number of variables had missing, invalid (e.g. #DIV/0!) and NA values. Finally, some variables did not seem to be related to the “classe” variable that had to be predicted. It was therefore necessary to clean the data before creating the models.

```
str( training[ ,1:16] )
```

```
## 'data.frame':    19622 obs. of  16 variables:
##  $ X                    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ user_name            : Factor w/ 6 levels "adelmo","carlitos",..: 2 2
2 2 2 2 2 2 2 2 ...
##  $ raw_timestamp_part_1: int  1323084231 1323084231 1323084231 132308423
2 1323084232 1323084232 1323084232 1323084232 1323084232 1323084232 ...
##  $ raw_timestamp_part_2: int  788290 808298 820366 120339 196328 304277 3
68296 440390 484323 484434 ...
##  $ cvtd_timestamp       : Factor w/ 20 levels "02/12/2011 13:32",..: 9 9
9 9 9 9 9 9 9 9 ...
##  $ new_window           : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1
1 1 ...
##  $ num_window           : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ roll_belt            : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.4
3 1.45 ...
##  $ pitch_belt           : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.1
6 8.17 ...
##  $ yaw_belt             : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4
-94.4 -94.4 -94.4 ...
##  $ total_accel_belt     : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ kurtosis_roll_belt   : Factor w/ 397 levels "","-0.016850",..: 1 1 1 1
1 1 1 1 1 1 ...
##  $ kurtosis_picth_belt : Factor w/ 317 levels "","-0.021887",..: 1 1 1 1
1 1 1 1 1 1 ...
##  $ kurtosis_yaw_belt    : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1
1 1 1 ...
##  $ skewness_roll_belt   : Factor w/ 395 levels "","-0.003095",..: 1 1 1 1
1 1 1 1 1 1 ...
##  $ skewness_roll_belt.1: Factor w/ 338 levels "","-0.005928",..: 1 1 1 1
1 1 1 1 1 1 ...
```

Data Exploration and Cleaning

At the data cleaning stage, the following points were addressed:

- **Identify and get rid of NA values** - By looking at the training data and applying the *is.na()* function column-wise it became apparent that a number of variables had a high frequency of NA values (98% of the observations) while other variables had valid values for each observation. The former were mainly summary statistics of primary variables such as mean, standard deviation, sample minimum and maximum, which could be derived from the primary variables in the dataset anyway. Therefore it was decided not to perform any data imputation and to remove the variables / columns with a high frequency of NA values. Specifically, 67 variables were pruned from the dataset at this stage.
- **Identify and get rid of blanks** - Similarly to the previous point, it was noticed that 33 variables were missing a value for 98% of the observations. Again, these variables were removed from the dataset and no imputation was performed.

- **Remove variables that are not related to the variable to be predicted** - After manual inspection, a total of seven variables were removed as they did not seem to bear any relationship to the “classe” variable that neede to be predicted. These were: “X”, “user_name”, “raw_timestamp_part_1”, “raw_timestamp_part_2”, “cvtd_timestamp”, “new_window”, “num_window”.

The code chunks below show what was performed at the data cleaning stage.

```
# Find na values and get rid of them
nasPerColumn <- colSums( is.na( training ) )
trainingClean <- training[ ,nasPerColumn==0]
```

```
# Find blanks (empty cells) and get rid of them
blanksPerColumn <- colSums( trainingClean == "" )
trainingClean <- trainingClean[ ,blanksPerColumn==0]
```

```
# Get rid of predictors that are not relevant / related to what we want to p
redict
colToBeDeleted <- c("X", "user_name", "raw_timestamp_part_1", "raw_timestamp
_part_2", "cvtd_timestamp", "new_window", "num_window" )
trainingClean <- trainingClean[ ,!names(trainingClean) %in% colToBeDeleted]
```

After the above cleaning operations the training dataset had a total of 19622 complete observations and 53 variables: 52 independent variables (potential predictors) and one dependent variable (the variable “classe” that needed to be predicted).

```
dim( trainingClean )
```

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

One last thing that was explored at this stage was the correlation between the 52 potential predictors in order to see if it was possible to get rid of redundant variables and reduce the size of the dataset any further. A correlation cut-off of 80% showed that 22 of the 52 independent variables were highly correlated.

```
correlations <- abs( cor( trainingClean[ ,-53] ) )
diag( correlations ) <- 0
highlyCorrelated <- which( correlations > 0.8, arr.ind = TRUE )
highlyCorrelated
```

```
##                   row col
## yaw_belt            3   1
## total_accel_belt    4   1
## accel_belt_y        9   1
## accel_belt_z       10   1
## accel_belt_x        8   2
## magnet_belt_x      11   2
## roll_belt           1   3
## roll_belt           1   4
## accel_belt_y        9   4
## accel_belt_z       10   4
## pitch_belt          2   8
## magnet_belt_x      11   8
## roll_belt           1   9
## total_accel_belt    4   9
## accel_belt_z       10   9
## roll_belt           1  10
## total_accel_belt    4  10
## accel_belt_y        9  10
## pitch_belt          2  11
## accel_belt_x        8  11
## gyros_arm_y        19  18
## gyros_arm_x        18  19
## magnet_arm_x       24  21
## accel_arm_x        21  24
## magnet_arm_z       26  25
## magnet_arm_y       25  26
## accel_dumbbell_x   34  28
## accel_dumbbell_z   36  29
## gyros_dumbbell_z   33  31
## gyros_forearm_z    46  31
## gyros_dumbbell_x   31  33
## gyros_forearm_z    46  33
## pitch_dumbbell     28  34
## yaw_dumbbell       29  36
## gyros_forearm_z    46  45
## gyros_dumbbell_x   31  46
## gyros_dumbbell_z   33  46
## gyros_forearm_y    45  46
```

```
length( unique( highlyCorrelated[ ,1] ) )
```

```
## [1] 22
```

This could potentially lead to further pruning of the training dataset by eliminating 13 redundant predictors as highlighted in the following code chunk.

```
names( trainingClean[ ,c(3,4,9,10,8,11,19,24,26,34,36,33,45)] )
```

```
##  [1] "yaw_belt"          "total_accel_belt" "accel_belt_y"
##  [4] "accel_belt_z"      "accel_belt_x"     "magnet_belt_x"
##  [7] "gyros_arm_y"       "magnet_arm_x"     "magnet_arm_z"
## [10] "accel_dumbbell_x" "accel_dumbbell_z" "gyros_dumbbell_z"
## [13] "gyros_forearm_y"
```

However it was decided to first fit the machine learning models to the 53 variables *trainingClean* dataset and leave the *trainingSuperClean* 40 variables dataset for further exploration.

Data Partitioning

Before fitting any models, the *trainigClean* dataset was partitioned using the caret package as follows:

- 60% for training the models (*tTraining* dataset): 11776 observations.
- 40% for testing the model (*tTesting* dataset): 7846 observations.

```
library( caret )
inTrain <- createDataPartition( trainingClean$classe, p = 0.6 )[[1]]
tTraining <- trainingClean[inTrain, ]
tTesting <- trainingClean[-inTrain, ]
```

In this way it was possible to estimate the out-of-sample error for the models before using them to predict the value of the variable “classe” of the testing dataset provided as part of this project. That is, the models were trained on the *tTraining* dataset and then used to predict the “classe” variables in the *tTesting* datset. By looking at the discrepancy between the predicted values and the real values of “classe” variable in *tTesting* it was possible to estimate the out-of-sample error of the models. The in-sample error was instead computed by looking at the discrepancy between the predicted value and the actual value of the variable “classe” in the *tTraining* dataset.

Model Fitting

Seven different models were fitted to the *tTraining* dataset and then assessed using the *tTesting* dataset. The explored models used different statistical techniques and tuning parameters as summarised below. Fitting times for each model were recorded.

- **modFit_LDA** Linear Discriminant Analysis model with *caret* default parameters and settings. Fitting time: almost instant.
- **modFit_PCA_LDA** Linear Discriminant Analysis model with *caret* default parameters and settings and Principal Component Pre-Processing. Fitting time almost instant.
- **modFit_RFDefault** Random Forests model with *caret* default parameters and settings. Fitting time: 1 h and 15 minutes.
- **modFit_RFCV10** Random Forests model with 10-fold cross-validation. Fitting time: 28 minutes.
- **modFit_RF50Trees** Random Forests model with only 50 trees (by default the *caret* package uses 500 trees). Fitting time: 8 minutes.
- **modFit_RF50TreesCV10** Random Forests model with 10-fold cross-validation and only 50 trees. Fitting time: 3 minutes
- **modFit_GBM** Gradient Boosting model with *caret* default settings. Fitting time: 30 minutes.

Regarding the Linear Discriminant Analysis with Principal Component Analysis pre-processing, the rationale behind was that the dataset had a 22 predictors that were highly correlated with each other and therefore the Principal Component Analysis pre-processing step could potentially help to reduce the number of predictors needed whilst capturing most of the variability in the data.

```
seed <- 1234
```

```
# Linear Discriminant Analysis
set.seed( seed )
modFit_LDA <- train( classe ~ ., method = "lda", data = tTraining )
predTTest_LDA <- predict( modFit_LDA, tTesting )
confusionMatrix( tTesting$classe, predTTest_LDA )

# Pre-process with PCA
preProc <- preProcess( tTraining, method = "pca", thresh = 0.80 )
trainPCA <- predict( preProc, tTraining )
set.seed( seed )
modFit_PCA_LDA <- train( classe ~ ., method = "lda", data = trainPCA )

# Linear Discriminant Analysis with PCA Pre-Processing
set.seed( seed )
testPCA <- predict( preProc, tTesting )
predTTest_PCA_LDA <- predict( modFit_PCA_LDA, testPCA )
confusionMatrix( tTesting$classe, predTTest_PCA_LDA )
```

```
# Default Random Forest
ptm <- proc.time()
set.seed( seed )
modFit_RFDefault <- train( classe ~ ., data = tTraining, method = "rf" )
print( "Time to fit the default Random Forest model: " )
proc.time() - ptm

predTTest_RFDefault <- predict( modFit_RFDefault, tTesting )
confusionMatrix( tTesting$classe, predTTest_RFDefault )

# Random Forest with 10-fold cv
trainControl_cv10 <- trainControl( method = "cv", number = 10 )

ptm <- proc.time()
set.seed( seed )
modFit_RFCV10 <- train( classe ~ ., data = tTraining, method = "rf", trContr
ol = trainControl_cv10 )
print( "Time to fit the Random Forest model with 10-fold cv: " )
proc.time() - ptm

predTTest_RFCV10 <- predict( modFit_RFCV10, tTesting )
confusionMatrix( tTesting$classe, predTTest_RFCV10 )

# Random Forest with only 50 trees
ptm <- proc.time()
set.seed( seed )
modFit_RF50Trees <- train( classe ~ ., data = tTraining, method = "rf", ntre
e = 50 )
print( "Time to fit the Random Forest model with only 50 trees: " )
proc.time() - ptm

predTTest_RF50Trees <- predict( modFit_RF50Trees, tTesting )
confusionMatrix( tTesting$classe, predTTest_RF50Trees )

# Random Forest with only 50 trees with 10-fold cv
ptm <- proc.time()
set.seed( seed )
modFit_RF50TreesCV10 <- train( classe ~ ., data = tTraining, method = "rf",
ntree = 50, trControl = trainControl_cv10 )
print( "Time to fit the Random Forest model with only 50 trees and 10-fold C
V: " )
proc.time() - ptm

predTTest_RF50TreesCV10 <- predict( modFit_RF50TreesCV10, tTesting )
confusionMatrix( tTesting$classe, predTTest_RF50TreesCV10 )
```

```
# Gradient Boosting
ptm <- proc.time()
set.seed( seed )
modFit_GBM <- train( classe ~ ., method = "gbm", data = tTraining )
print( "Time to fit the default Gradient Boosting model: " )
proc.time() - ptm

predTTest_GBM <- predict( modFit_GBM, tTesting )
confusionMatrix( tTesting$classe, predTTest_GBM )
```

Model Assessment and Error Estimation

The seven models described in the previous section were used to predict the “classe”. Their in-sample accuracy and erros are an indication of how well they predicted the “classe” variable in the *tTrainig* dataset. On the other hand, an estimate of their out-of-sample accuracy and error could be given by assessing how well they predicted the “classe” variable in the *tTesting* dataset.

The accuracy of a model could be determined by simply inspecting the model object or by calculating it manually.

```
modFit_LDA
```

```
## Linear Discriminant Analysis
##
## 11776 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 11776, 11776, 11776, 11776, 11776, 11776, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.6984779  0.6184473
##
##
```

```
sum( predict( modFit_LDA , tTraining ) == tTraining$classe ) / length( tTrai
ning$classe )
```

```
## [1] 0.708305
```

Similarly, the out-of-sample accuracy could be determined by inspecting the confusion matrix object by calculating it manually.

```
predTTest_LDA <- predict( modFit_LDA, tTesting )
cm_LDA <- confusionMatrix( tTesting$classe, predTTest_LDA )
cm_LDA$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNul
l
##   7.035432e-01   6.250329e-01   6.932999e-01   7.136351e-01   2.922508e-0
1
## AccuracyPValue  McnemarPValue
##   0.000000e+00   2.703327e-79
```

```
sum( predict( modFit_LDA, tTesting ) == tTesting$classe) / length( tTesting
$classe )
```

```
## [1] 0.7035432
```

The table below summarises the performance for the seven models, in terms of in-sample accuracy, in-sample error, out-of-sample accuracy, out-of-sample error and run-time required to fit the model. The accuracy values reported are taken from the model and confusion matrix objects respectively.

```
##              modelName    inSAcc    inSErr   outSAcc   outSErr    runTime
## 1           modFit_LDA 0.6984779 0.3015221 0.7069845 0.2930155    < 1 min
## 2      modFit_PCA_LDA2 0.4626796 0.5373204 0.4734897 0.5265103    < 1 min
## 3     modFit_RFDefault 0.9852415 0.0147585 0.9917155 0.0082845 1 h 15 min
## 4        modFit_RFCV10 0.9904895 0.0095105 0.9918430 0.0081570     28 min
## 5     modFit_RF50Trees 0.9824113 0.0175887 0.9903135 0.0096865      8 min
## 6 modFit_RF50TreesCV10 0.9896402 0.0103598 0.9898037 0.0101963      3 min
## 7           modFit_GBM 0.9553872 0.0446128 0.9599796 0.0400204     30 min
```

Based from the results it is possible to see that the best models are the ones generated using the Random Forest algorithm. In particular, the Random Forest model created using 10-fold Cross Validation appears to be the best, also in terms of accuracy - run-time trade-off. On the other hand, the worst model is the one generated using the Principal Component Analysis pre-processing step and the Linear Discriminant Analysis algorithm. These results could suggest that the problem has a non-linear nature.

One last aspect that was explored was the fitting of the predictive models to the *trainingSuperClean* dataset with only 40 variables (see the Data Cleaning section above). The *trainingSuperClean* dataset was partitioned in a *tTrainingSC* dataset containing 60% of the observations and a *tTestingSC* dataset containig the remaining 40% of the observations. The four Random Forest models as specified above were then trained by using the *tTrainingSC* dataset and used to predict the “classe” variable in the *tTestingSC* dataset. The results are summarised in the table below.

```
### Random Forests without highly correlated variables

trainingSuperClean <- trainingClean[ ,-c(3,4,9,10,8,11,19,24,26,34,36,33,4
5)]

inTrain <- createDataPartition( trainingSuperClean$classe, p = 0.6 )[[1]]
tTrainingSC <- trainingSuperClean[inTrain, ]
tTestingSC <- trainingSuperClean[-inTrain, ]

# Default Random Forest
ptm <- proc.time()
set.seed( seed )
modFit_RFDefaultSC <- train( classe ~ ., data = tTrainingSC, method = "rf" )
print( "Time to fit the default SC Random Forest model: " )
proc.time() - ptm

predTTest_RFDefaultSC <- predict( modFit_RFDefaultSC, tTestingSC )
confusionMatrix( tTestingSC$classe, predTTest_RFDefaultSC )

# Random Forest with 10-fold cv
ptm <- proc.time()
set.seed( seed )
modFit_RFCV10SC <- train( classe ~ ., data = tTrainingSC, method = "rf", trC
ontrol = trainControl_cv10 )
print( "Time to fit the SC Random Forest model with 10-fold cv: " )
proc.time( "59 min", ) - ptm

predTTest_RFCV10SC <- predict( modFit_RFCV10SC, tTestingSC )
confusionMatrix( tTestingSC$classe, predTTest_RFCV10SC )

# Random Forest with only 50 trees
ptm <- proc.time()
set.seed( seed )
modFit_RF50TreesSC <- train( classe ~ ., data = tTrainingSC, method = "rf",
ntree = 50 )
print( "Time to fit the SC Random Forest model with only 50 trees: " )
proc.time() - ptm

predTTest_RF50TreesSC <- predict( modFit_RF50TreesSC, tTestingSC )
confusionMatrix( tTestingSC$classe, predTTest_RF50Trees )

# Random Forest with only 50 trees with 10-fold cv
ptm <- proc.time()
set.seed( seed )
modFit_RF50TreesCV10SC <- train( classe ~ ., data = tTrainingSC, method = "r
f", ntree = 50, trControl = trainControl_cv10 )
print( "Time to fit the SC Random Forest model with only 50 trees and 10-fol
d CV: " )
proc.time() - ptm

predTTest_RF50TreesCV10SC <- predict( modFit_RF50TreesCV10SC, tTestingSC )
confusionMatrix( tTestingSC$classe, predTTest_RF50TreesCV10SC )
```

The results are summarised in the table below.

```
##                modelName    inSAcc    inSErr   outSAcc   outSErr runTime
## 1     modFit_RFDefaultSC 0.9829314 0.0170686 0.9890390 0.0109610  51 min
## 2        modFit_RFCV10SC 0.9876871 0.0123129 0.9892939 0.0107061  19 min
## 3     modFit_RF50TreesSC 0.9806464 0.0193536 0.9903135 0.0096865   2 min
## 4 modFit_RF50TreesCV10SC 0.9852241 0.0147759 0.9884017 0.0115983   2 min
```

These results indicate that the performance of the four Random Forest algorithms trained on the training dataset with only 40 variables (*tTrainingSC*) are comparable to the performance of the equivalent models trained on the bigger training dataset with 53 variables (*tTraining*).

Conclusions

The variable “classe” in the final testing dataset (*testing*, read straight from the pml-testing.csv file) was predicted using the two best models trained in the 53 variable training dataset and the 40 variables training dataset respectively, namely the 10-fold cross-validation Random Forest model (**modFit_RFCV10**) and the Random Forest model with only 50 trees (**modFit_RF50TreesSC**).

```
predict( modFit_RFCV10, testing )
```

```
## Warning: package 'randomForest' was built under R version 3.2.5
```

```
##  [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
```

```
predict( modFit_RF50TreesSC, testing )
```

```
##  [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
```

The predictions are concordant and were used as the final answer for this assignment.