Practical Machine Learning Assignment

Marcos A. Santos

2022-07-04

Overview

The task developed in this document is for the completion of the Practical Machine Learning Course Assignment, part of Coursera's Data Science Certification by Johns Hopkins University.

Background Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement – a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website here: http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Requested demonstrations and actions

1. The main objective is to predict how correct, on a scale of 5 levels, one would perform a given exercise by observing data from the devices described above using prior training data.
2. Create a report explaining how the model was built and why a certain technique was chosen to do it.
3. Use the model to predict 20 test cases and submit the results to the automated grading.
4. Create a Github repo and upload a compiled HTML file and a R Markdown of the report. Constrain the text to a maximum of 2000 words and 5 figures.

Given conditions || assumptions

1. The data for training and testing are pre-determined.
2. There is no or little prior knowledge of the specific theory about the inner workings of the process.
3. The EDA phase is allowed to normalize and judge all training features.
4. There is no limitation on the needed processing power for the training nor the maximum latency to predict.
5. It should be used the techniques taught in classes.

Task 1 - Load the needed libraries and work data

The following environment was chosen to accomplish the objective: RStudio 2022.02.1 Build 461, R version 4.0.4 (2021-02-15), caret 6.0.92, rpart 4.1.15, rpart.plot 3.1.1, corrplot 0.92, randomForest 4.6.14.

The random seed used to reference our stats was set to **88570**

The traning and test data used in this project are available at <https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv> and <https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv> respectively and kindly provided by: <http://groupware.les.inf.puc-rio.br/har>. The loaded dataframes have the following structure :

Dataframe	Num. of Observations	Num. of Variables
training_raw	19622	160
testing_raw	20	160

The target to predict is the variable `classe`.

Task 2 - Feature Engineering

First, by observing the test dataframe we may find columns that are empty so it's safe to remove them. This reduced our variables amount to :

Dataframe	Num. of Observations	Num. of Variables
training_clean	19622	60
testing_clean	20	60

Now, assuming that we are using only the sensors data to learn, let's remove other non relevant columns to reduce noise (index, user name and temporal references), this will bring our variables to :

Dataframe	Num. of Observations	Num. of Variables
training_clean	19622	55
testing_clean	20	55

The target variable `classe` is originally of type character. Let's change to a factor in order to easy our model and cross validation algorithms .

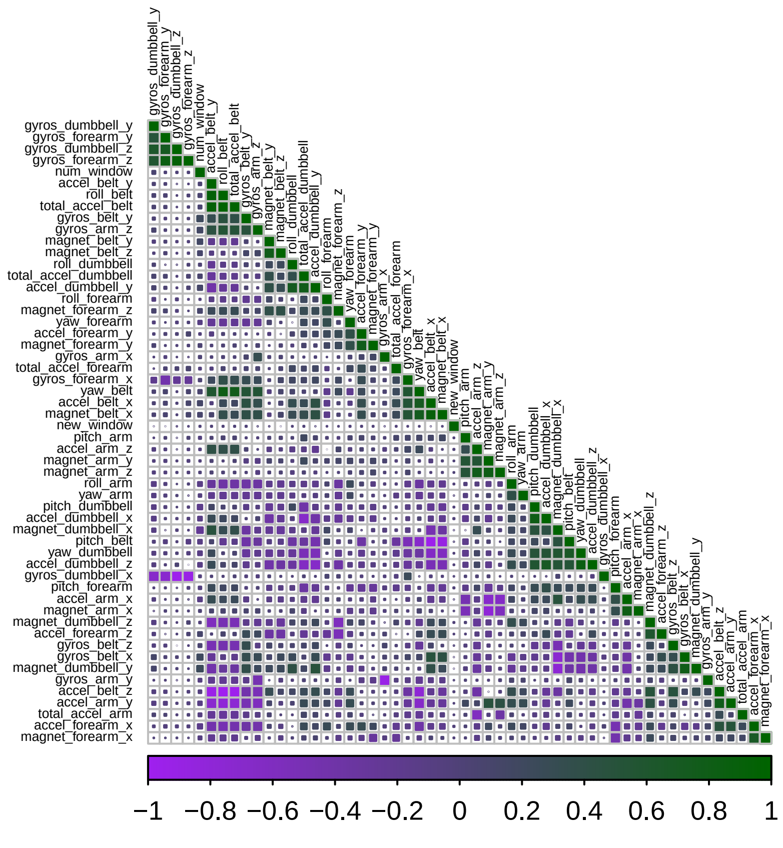
```
##    A    B    C    D    E
## 5580 3797 3422 3216 3607
```

Task 3 - EDA

Is there any noticeable correlations with our target variable `classe` ? - What `stats cor` can tell us ?

```
##              Var1 Var2       Freq
## 14 magnet_belt_y    A -0.2903491
## 26  magnet_arm_x    A  0.2959636
## 27  magnet_arm_y    A -0.2566702
## 43 pitch_forearm    A  0.3438258
```

Hard to tell there is a strong one, the best one is close to 0.30. Now let's take a look on the overall features correlation :



It looks like some features are quite correlated with each other. We may try to reduce them by PCA should we need help due that our final result doesn't meet the specifications or we have time and resources to do some research in order to improve performance. For now, we're restricted to 2000 words.

Task 3 - Implementation Setup - Partitioning

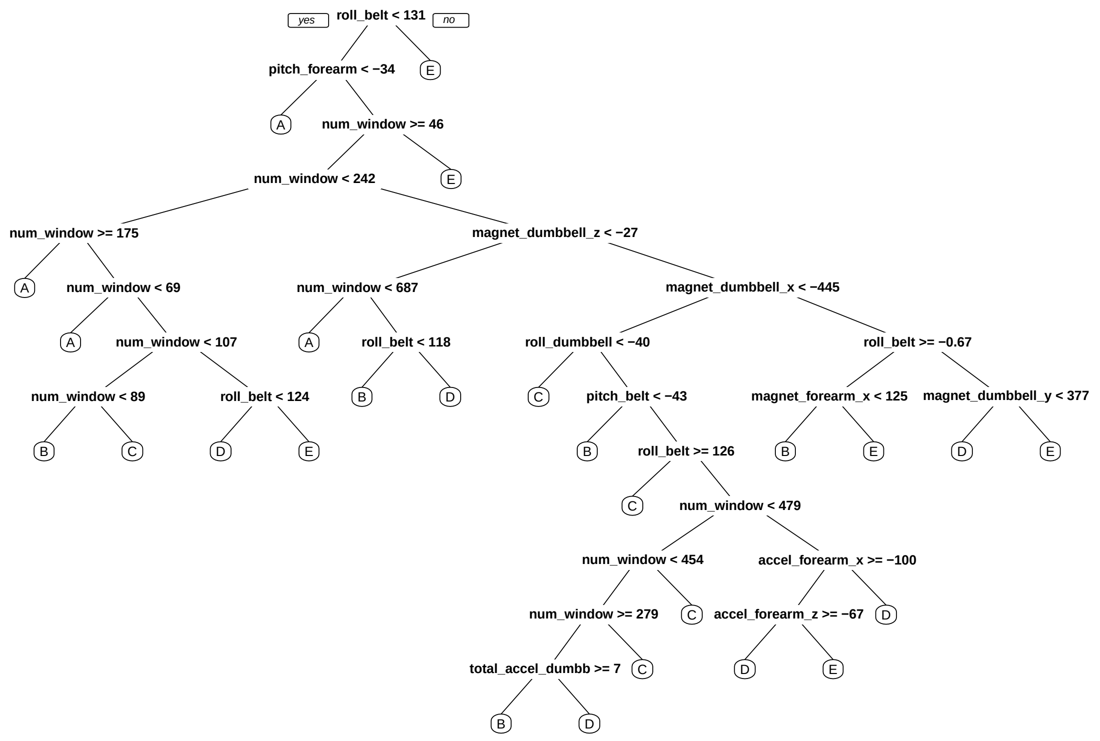
Since the test set is reserved to the ultimate verification, we will extract the validation set from the training. The split will be the common 3/4. Therefore we will end up with the following sets:

Dataframe	Num. of Observations
training	14718
validation	4904
test	20

Data Modelling

Task 4 - Baseline Approach -> Decision Tree

Since we're trying to classify the exercises, the obvious baseline algorithm is the **Decision Tree** and here is the structure he found :



Now, the performance evaluation using the validation set gives us the following figures:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1160  141    0   79   15
##          B   60  739   52   37   61
##          C    0   57  713   73   12
##          D    8   62   35  638   61
##          E    4   41    3   56  797
##
## Overall Statistics
```

```
##
##                Accuracy : 0.8252
##                  95% CI : (0.8143, 0.8358)
##     No Information Rate : 0.2512
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7799
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9416   0.7106   0.8879   0.7225   0.8425
## Specificity            0.9360   0.9457   0.9654   0.9587   0.9737
## Pos Pred Value         0.8315   0.7787   0.8339   0.7935   0.8846
## Neg Pred Value         0.9795   0.9239   0.9778   0.9402   0.9628
## Prevalence             0.2512   0.2121   0.1637   0.1801   0.1929
## Detection Rate         0.2365   0.1507   0.1454   0.1301   0.1625
## Detection Prevalence   0.2845   0.1935   0.1743   0.1639   0.1837
## Balanced Accuracy      0.9388   0.8281   0.9266   0.8406   0.9081
```

Looks like there is a big room to improvement. Let's register some parameters to further benchmarks:

Performance Gauge	Result
Accuracy	82.52 %
Out-of_Sample Error (est)	17.48 %
Time to train	5.63 sec.
Latency to predict	0.084 sec.

Task 5 - Improving the trees -> Random Forest

The **Random Forrest** algorithm in R using a **5-Fold cross validation** is knew to perform well due his best procedures to manage outliers and the automatic selection of variables, so we should get better results. It would be informative to benchmark the procedures with number of trees around 50 just to see how would be if working in dynamic industrial control environments (process response is a real concern). For brevity here let's do only for 10 and 100 trees. Here are the models :

For 100 trees :

```
## Random Forest
##
## 14718 samples
##    54 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 11774, 11775, 11773, 11774, 11776
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9930698  0.9912334
##   28    0.9971463  0.9963904
##   54    0.9942923  0.9927801
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 28.
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    0    0    0    0
##          B    1  948    0    0    0
##          C    0    2  853    0    0
##          D    0    0    2  802    0
##          E    0    0    0    1  900
##
## Overall Statistics
##
##                Accuracy : 0.9988
##                  95% CI : (0.9973, 0.9996)
##     No Information Rate : 0.2847
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9985
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9993   0.9979   0.9977   0.9988   1.0000
## Specificity            1.0000   0.9997   0.9995   0.9995   0.9998
## Pos Pred Value         1.0000   0.9989   0.9977   0.9975   0.9989
## Neg Pred Value         0.9997   0.9995   0.9995   0.9998   1.0000
## Prevalence             0.2847   0.1937   0.1743   0.1637   0.1835
## Detection Rate         0.2845   0.1933   0.1739   0.1635   0.1835
## Detection Prevalence   0.2845   0.1935   0.1743   0.1639   0.1837
## Balanced Accuracy      0.9996   0.9988   0.9986   0.9991   0.9999
```

And for only 10 trees :

```
## Random Forest
##
## 14718 samples
##    54 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 11775, 11776, 11773, 11774, 11774
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9813842  0.9764488
##   28    0.9927301  0.9908044
##   54    0.9934776  0.9917495
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 54.
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1394    1    0    0    0
##          B    1  944    4    0    0
##          C    0    3  852    0    0
##          D    1    5    8  790    0
##          E    0    0    0    1  900
##
## Overall Statistics
##
##                Accuracy : 0.9951
##                  95% CI : (0.9927, 0.9969)
##     No Information Rate : 0.2847
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9938
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9986   0.9906   0.9861   0.9987   1.0000
## Specificity            0.9997   0.9987   0.9993   0.9966   0.9998
## Pos Pred Value         0.9993   0.9947   0.9965   0.9826   0.9989
## Neg Pred Value         0.9994   0.9977   0.9970   0.9998   1.0000
## Prevalence             0.2847   0.1943   0.1762   0.1613   0.1835
## Detection Rate         0.2843   0.1925   0.1737   0.1611   0.1835
## Detection Prevalence   0.2845   0.1935   0.1743   0.1639   0.1837
## Balanced Accuracy      0.9991   0.9946   0.9927   0.9977   0.9999
```

```
Here are some important values to shed some light :
```

Performance Gauge	10 trees	100 trees
Accuracy @ mtry	99.51 %	99.88 %
Out-of_Sample Error (est)	0.49 %	0.12 %
Time to train	37.18 sec.	310.214 sec.
Latency to predict	0.096 sec.	0.607 sec.

The Random Forrest produced models that are at first glance doing a good job on predicting the exercises. There is no concerns about accuracy and we will probably get a good AUC (not shown, again we should restrict to 2000 words)

However, the model with 100 trees is useless on control loops (training of more than 300 sec is too long and latency of 600ms is barely enough in most of the systems). The model with 10 trees performs better on latency than a single decision tree and is somehow acceptable with his 90 ms. His training time can be upgraded by maybe using what was suggested in the EDA phase (reducing the dimension of the correlated x,y,z accelerometers axes by PCA and therefore the number of features) but this has to be based on knowledge of the sensors intrinsics.

Task 6 - Predictions from Test Data

Here are the results that our model predicted from the provided 20 samples:

```
##  [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 20 files with result for each question of the assignment are available at <https://github.com/msantrax/coursera_johns_hopkins/blob/r

References :

APPENDIX - CODE

```r
# Load the needed libraries
library(caret)
library(rpart)
library(rpart.plot)
library(corrplot)
library(randomForest)

# Normalize stats random
set.seed(88570)

# Load the work data locally
training_raw <- read.csv("pml-training.csv")
testing_raw <- read.csv("pml-testing.csv")
```

```r
# Clean empty features
emptymap <- (colSums(is.na(testing_raw)) == 0)
training_clean <- training_raw[, emptymap]
testing_clean <- testing_raw[, emptymap]
rm(emptymap)
```

```r
# Clean empty features
training_clean$classe <- factor(training_clean$classe)
summary(training_clean$classe)
```

```r
# Detect correlation with target
training_corr <- data.frame(data.matrix(training_clean))
classe_idx <- which(names(training_corr) == "classe")
correlations <- cor(training_corr[, -classe_idx], as.numeric(training_corr$classe))
best_correlations <- subset(as.data.frame(as.table(correlations)), abs(Freq)>0.25)
best_correlations
```

```r
# Detect correlation among features
corrplot(
  cor(training_corr[, -length(names(training_clean))]),
  method = "square",
  type = "lower",
  order = "hclust",
  tl.col = "black",
  tl.cex = 0.4,
  col = colorRampPalette(c("purple", "dark green"))(200)
)
```

```r
# Create the validation set
inTrain <- createDataPartition(training_clean$classe, p = 3/4, list = FALSE)
validation <- training_clean[-inTrain, ]
training <- training_clean[inTrain, ]
```

```r
# Model by Decision Tree
start <- proc.time()
model_dtree <- rpart(classe ~ ., data = training, method = "class")
prp(model_dtree)
model_dtree_time = proc.time() - start
model_dtree_time = model_dtree_time[3]
```

```r
# Evaluate the decision tree
start <- proc.time()
predict_dtree <- predict(model_dtree, validation, type = "class")
confusionMatrix(validation$classe, predict_dtree)
predict_dtree_time = proc.time() - start
predict_dtree_time = predict_dtree_time[3]
```

```r
# Get the accuracy and estimated out-of-sample
dtree_accuracy <- postResample(predict_dtree, validation$classe)
dtree_accuracy <- sprintf("%6.2f %%", dtree_accuracy[1]*100)
dtree_ose <- 1 - as.numeric(confusionMatrix(validation$classe, predict_dtree)$overall[1])
dtree_ose <- sprintf("%6.2f %%", dtree_ose*100)
```

```r
## Edition stub - load a pre-processed model from archived file
model_rf100 <- readRDS("rf_100.rds")
modeling_time100 = c(309.021, 0.670, 310.214, 0, 0)

# Generate the Random Forrest Model
# start <- proc.time()
# model_rf100 <- train(classe ~ ., data = training, method = "rf", trControl = trainControl(method = "cv", 5), r

modeling_time100 = modeling_time100[3]
model_rf100

start <- proc.time()
predict_rf100 <- predict(model_rf100, validation)
confusionMatrix(validation$classe, predict_rf100)
predicting_time100 = proc.time() - start
predicting_time100 = predicting_time100[3]

# Get the accuracy and estimated out-of-sample
rf100_accuracy <- postResample(predict_rf100, validation$classe)
rf100_accuracy <- sprintf("%6.2f %%", rf100_accuracy[1]*100)
rf100_ose <- 1 - as.numeric(confusionMatrix(validation$classe, predict_rf100)$overall[1])
rf100_ose <- sprintf("%6.2f %%", rf100_ose*100)
```

```r
## Edition stub - load a pre-processed model from archived file
model_rf10 <- readRDS("rf_10.rds")
modeling_time10 = c(36.869, 0.240, 37.180, 0, 0)

# Generate the Random Forrest Model
# start <- proc.time()
# model_rf100 <- train(classe ~ ., data = training, method = "rf", trControl = trainControl(method = "cv", 5), r

modeling_time10 = modeling_time10[3]
model_rf10

start <- proc.time()
predict_rf10 <- predict(model_rf10, validation)
confusionMatrix(validation$classe, predict_rf10)
predicting_time10 = proc.time() - start
predicting_time10 = predicting_time10[3]

# Get the accuracy and estimated out-of-sample
rf10_accuracy <- postResample(predict_rf10, validation$classe)
rf10_accuracy <- sprintf("%6.2f %%", rf10_accuracy[1]*100)
rf10_ose <- 1 - as.numeric(confusionMatrix(validation$classe, predict_rf10)$overall[1])
rf10_ose <- sprintf("%6.2f %%", rf10_ose*100)
```

```r
# Show the results obtained by prediction over test data
pred_test = predict(model_rf100, testing_clean[, -length(names(testing_clean))])
n = length(pred_test)
for(i in 1:n){
  filename = paste0("Assignment/test_",i,".txt")
  write.table(pred_test[i],
              file = filename,
              quote = FALSE,
              row.names = FALSE,
              col.names = FALSE)
}

pred_test
```