Practical Machine Learning Project

Nelson H. Tejara

23 November 2018

Introduction

This document presents a way on how to build a predictive model for the Weight Lifting EXercises Dataset found [here] (http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)). The goal of this project is to determine the manner in which the exercise is performed. The random forest model was used by the author in order to develop the predictive model.

Data Acquisition and Cleaning

Firstly, the training and testing data has been downloaded. These data will be used for the training and validation of the predictive model. From the data obtained some variables were of factor class. Upon checking, these variable it contains numeric entries that represents their levels.

Some preliminary observations includes : * Some variables are useless (e.g. `levels(pml.data$kurtosis_yaw_belt)` -> “#DIV/0!”) * Some have actually a wide range of numeric values but also a “#DIV/0!” value.

```
pml.data <- read.csv("data/pml-training.csv", na.strings=c("NA",""))
# str(pml.data)
# check levels of factor variables to see those with useless levels and
# see which variables should be numeric
# The code below is used to take a look at the  factor variables levels
#for (colName in colnames(pml.data[ ,sapply(pml.data, class) == "factor"])) {
#  print(colName)
#  print(levels(pml.data[[colName]]))
#}
# Remove useless variables from the data set.
variables <- c(
  "X", "user_name", "raw_timestamp_part_1", "raw_timestamp_part_2",
  "kurtosis_yaw_belt", "skewness_yaw_belt", "amplitude_yaw_belt", "cvtd_timestamp",
  "kurtosis_yaw_dumbbell", "skewness_yaw_dumbbell", "amplitude_yaw_dumbbell",
  "kurtosis_yaw_forearm", "skewness_yaw_forearm", "amplitude_yaw_forearm"
)
pml.data <- pml.data[ , -which(names(pml.data) %in% variables)]


# Convert factor variables which are actually numeric, to numeric variables.
variables <- c(
  "kurtosis_roll_belt", "kurtosis_picth_belt", "skewness_roll_belt",
  "skewness_roll_belt.1", "max_yaw_belt", "min_yaw_belt",
  "kurtosis_roll_arm","kurtosis_picth_arm", "kurtosis_yaw_arm",
  "skewness_roll_arm","skewness_pitch_arm", "skewness_yaw_arm",
  "kurtosis_roll_dumbbell", "kurtosis_picth_dumbbell","skewness_roll_dumbbell",
  "skewness_pitch_dumbbell","max_yaw_dumbbell", "min_yaw_dumbbell",
  "kurtosis_roll_forearm","kurtosis_picth_forearm", "skewness_roll_forearm",
  "skewness_pitch_forearm", "max_yaw_forearm", "min_yaw_forearm"
)
for (variable in variables) {
  pml.data[[variable]] <- as.numeric(as.character(pml.data[[variable]]))
}
```

It was found out that there are many variable that contains NA values. These values will affect during the training of our predictive model.To do this, a model will be build with no NA values at all utilizing only those variables with no NA values. If this will not work, a technique for gap filling will be employed in order to cope the NA values (eg. k- nearest neighbor).

```
pml.data.naCounts <- colSums(sapply(pml.data, is.na))
pml.data.complete <- pml.data[,pml.data.naCounts == 0]
```

We have gone from 146 variables to 55 variables.

Training the model

The next step is to train the model. The data is split to training and testing set to follow the cross validation practices. As recommended from the course, it is 60/40 for training and testing.

```
library(caret)
```

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

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

```
set.seed(20052015) # Set a seed for reproducibility purposes
# Split the training set in two parts: 60/40. The first part will be used for
# train the model, and the second part will be used for validation of the model.
inTrain <- createDataPartition(pml.data.complete$classe, p=0.6, list = FALSE)
pml.data.train <- pml.data.complete[inTrain,]
pml.data.test <- pml.data.complete[-inTrain,]
```

Before fitting the model, checking must be done to see if the training contains numeric values that have no or close to zero variance.

```
nearZeroVar(pml.data.train[, sapply(pml.data.train, is.numeric)])
```

```
## integer(0)
```

Since there are no numeric variables with near zero variance, one cannot reduce the number of variables this way. Hence, we can continue building the model utilizing all the remaining variables in the training dataset.

```
library(caret)
modelFit <- train(classe ~ ., data = pml.data.train, method="rf", importance=TRUE)
```

Verify the mode

```
# Now test the model with the test data.
predictionResults <- confusionMatrix(pml.data.test$classe, predict(modelFit, pml.data.te
st))
predictionResults$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 2230    1    0    0    1
##          B    5 1513    0    0    0
##          C    0    6 1361    1    0
##          D    0    0   13 1272    1
##          E    0    0    0    7 1435
```

The total number of predictions is: `nrow(pml.data.test)` : 7846. This data is equal to the the sum of all items in the prediction matrix `sum(predictionResults$table)` : 7846. Only the diagonal represent counts of correct predicted results. As the predicted results were generated with data which was not used for testing, we can calculate the out-of-sample error rate by substracting the sum of the diagonal from the toal: `sum(predictionResults$table) - sum(diag(predictionResults$table))` : 35 out of 7846 (or 0.4460872%). In other words, the model succeeded quite well in predicting the activities on the training data set.

Predict “new” data

Now we have a model that seems to perform well, we use it to predict the verification data. Since we now have the model that performs quiet well, then we can use it to predict the verification data.

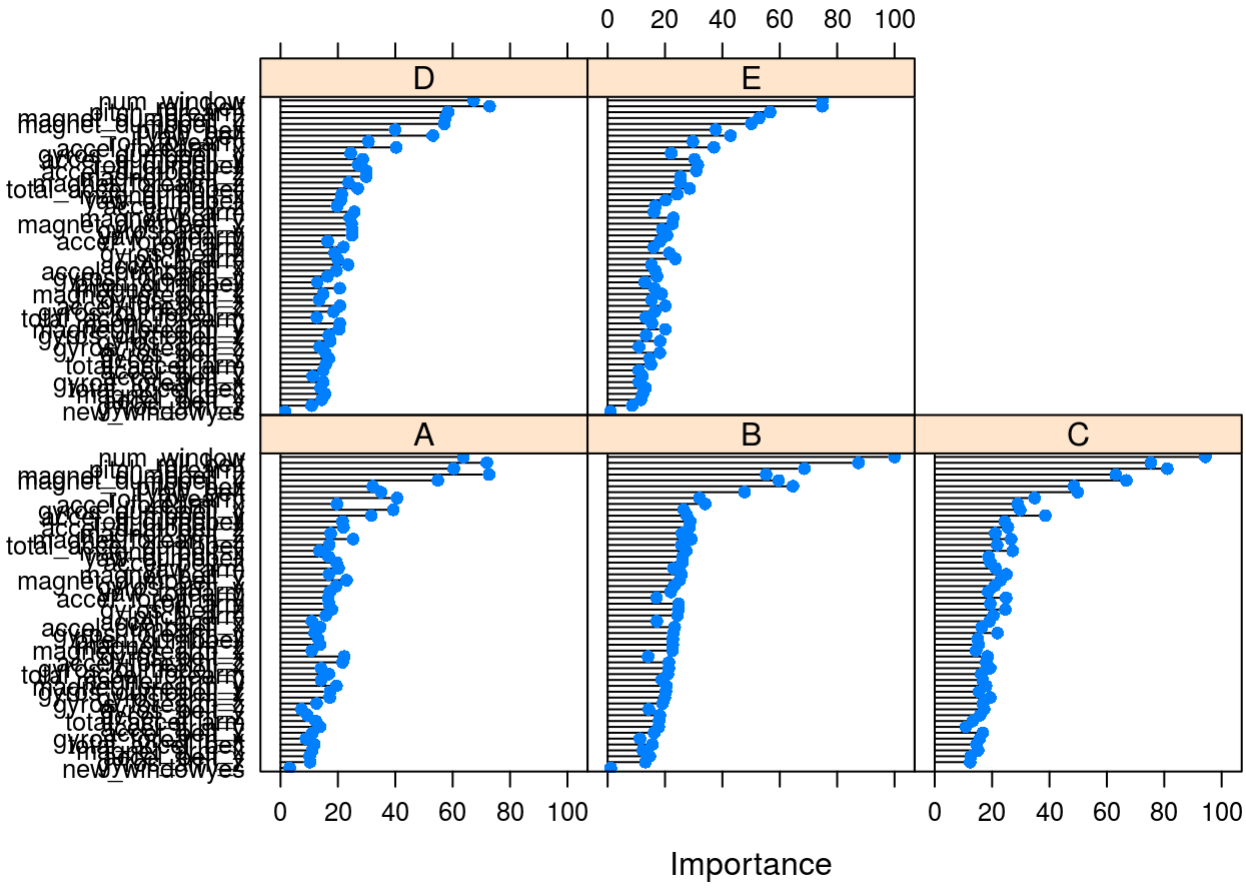
```
pml.verification <- read.csv("data/pml-testing.csv")
pml.verification <- pml.verification[, colnames(pml.verification) %in% colnames(pml.dat
a.train)]
predict(modelFit, pml.verification)
```

```
##  [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
```

Appendix:

The variables from `ncol pml.data.train` have been used. When we look at the variable importances in our model, we see a rather quick drop off.

```
variable.importances <- varImp(modelFit)
plot(variable.importances)
```



With this, one can see if he can train a model with lower number of variables having similar precision. To do this, one needs to get the row of sums of the variable importances and normalize their values.

```
variable.importances.sums <- sort(rowSums(variable.importances$importance), decreasing =
 T)
variable.importances.sums <- variable.importances.sums / sum(variable.importances.sums)
```

If we take the first 16 variables we have captured about 50 percent of the overall importances:

```
sum(variable.importances.sums[1:16])
```

```
## [1] 0.5122935
```

```
new.train.vars <- c("classe", names(variable.importances.sums[1:16]))
new.train.data <- pml.data.train[, colnames(pml.data.train) %in% new.train.vars]
new.test.data <- pml.data.test[, colnames(pml.data.train) %in% new.train.vars]
```

```
modelFit.reduced <- train(classe ~ ., data = new.train.data, method="rf", importance=TRU
E)
```

Using the new mode1, it can be use again to test the data.

```
confusionMatrix(new.test.data$classe, predict(modelFit.reduced, new.test.data))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2230    1    0    0    1
##          B    2 1515    1    0    0
##          C    0    6 1362    0    0
##          D    0    0    3 1282    1
##          E    0    0    0    7 1435
##
## Overall Statistics
##
##                Accuracy : 0.9972
##                  95% CI : (0.9958, 0.9982)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9965
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9991   0.9954   0.9971   0.9946   0.9986
## Specificity            0.9996   0.9995   0.9991   0.9994   0.9989
## Pos Pred Value         0.9991   0.9980   0.9956   0.9969   0.9951
## Neg Pred Value         0.9996   0.9989   0.9994   0.9989   0.9997
## Prevalence             0.2845   0.1940   0.1741   0.1643   0.1832
## Detection Rate         0.2842   0.1931   0.1736   0.1634   0.1829
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9994   0.9975   0.9981   0.9970   0.9988
```

As reflected, it was able to produce as good results as the first. The training of this model took about 30 minutes as compared to ~1.5 hours for the first model.