Human Activity Recognition

This project aims to apply Machine Learning Techniques to develop an algorithm to classify activities performed by different subjects on the dataset collected in the link below:

http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har

1. Setup

```
rm(list=ls())

library(caret)
library(readr)
library(dplyr)
library(ggplot2)
library(gridExtra)

setwd("C:/Users/cmffe/OneDrive - Vestas Wind Systems A S/Documents/R/Practical Machine Learning - Cours
```

2. Reading the dataset

```
HAR <- read_csv("pml-training.csv", col_names = TRUE)
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Warning: 182 parsing failures.
##  row               col expected  actual               file
## 2231 kurtosis_roll_arm a double #DIV/0! 'pml-training.csv'
## 2231 skewness_roll_arm a double #DIV/0! 'pml-training.csv'
## 2255 kurtosis_roll_arm a double #DIV/0! 'pml-training.csv'
## 2255 skewness_roll_arm a double #DIV/0! 'pml-training.csv'
## 2282 kurtosis_roll_arm a double #DIV/0! 'pml-training.csv'
## .... ................. ........ ....... ..................
## See problems(...) for more details.
```

```
final_test <- read_csv("pml-testing.csv", col_names = TRUE)
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

3. Performing Data Partition

```
set.seed(123)
inTrain <- createDataPartition(y = HAR$X1, p = 0.5, list = FALSE)
training <- HAR[inTrain,]
testing <- HAR[-inTrain,]
```

4. Looking for NA values and cleaning up the database

Checking for columns with over 80% of NA values and filtering them out.

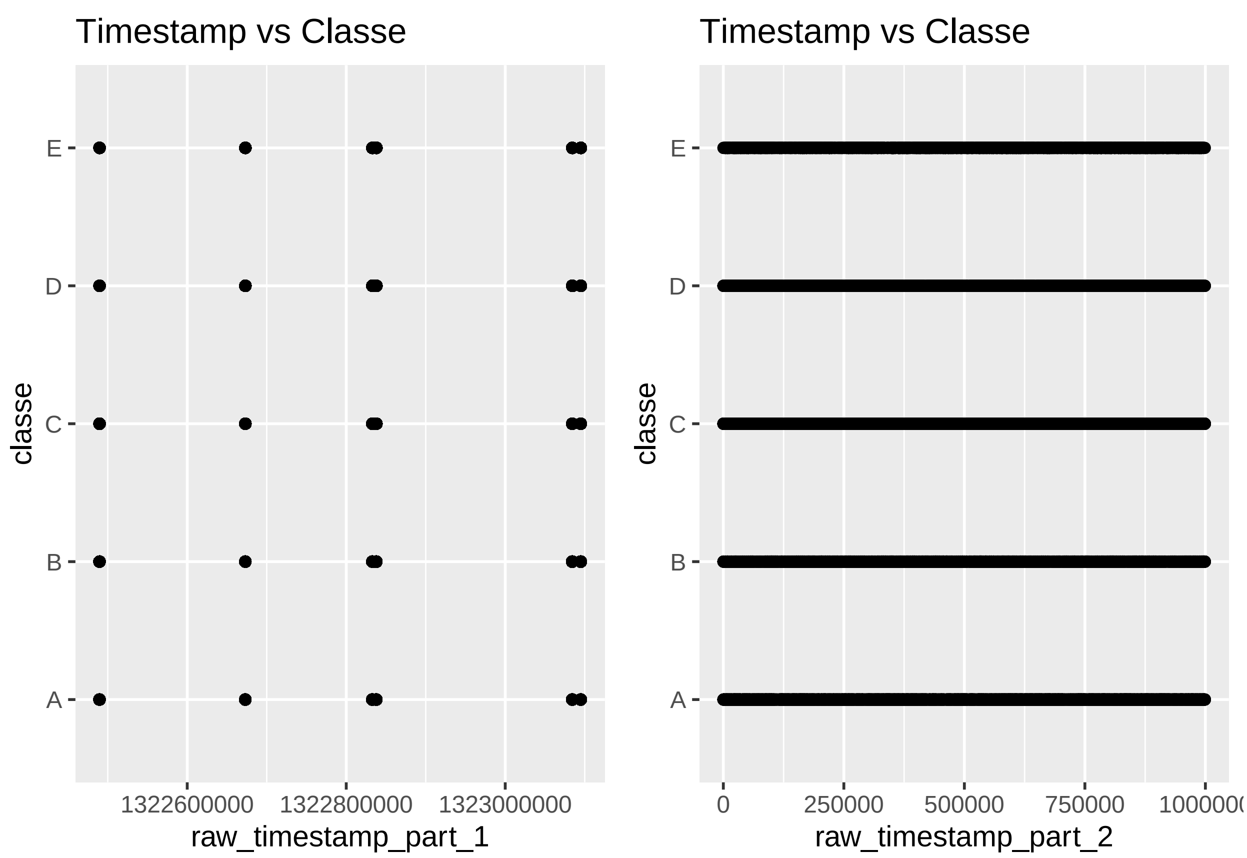
```
NAcols <- data.frame(variable = colnames(training), percentage_NA = colSums(is.na(training))/nrow(train
NAcols <- NAcols %>% filter(percentage_NA > 0.8)
count(NAcols)
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1   100
```

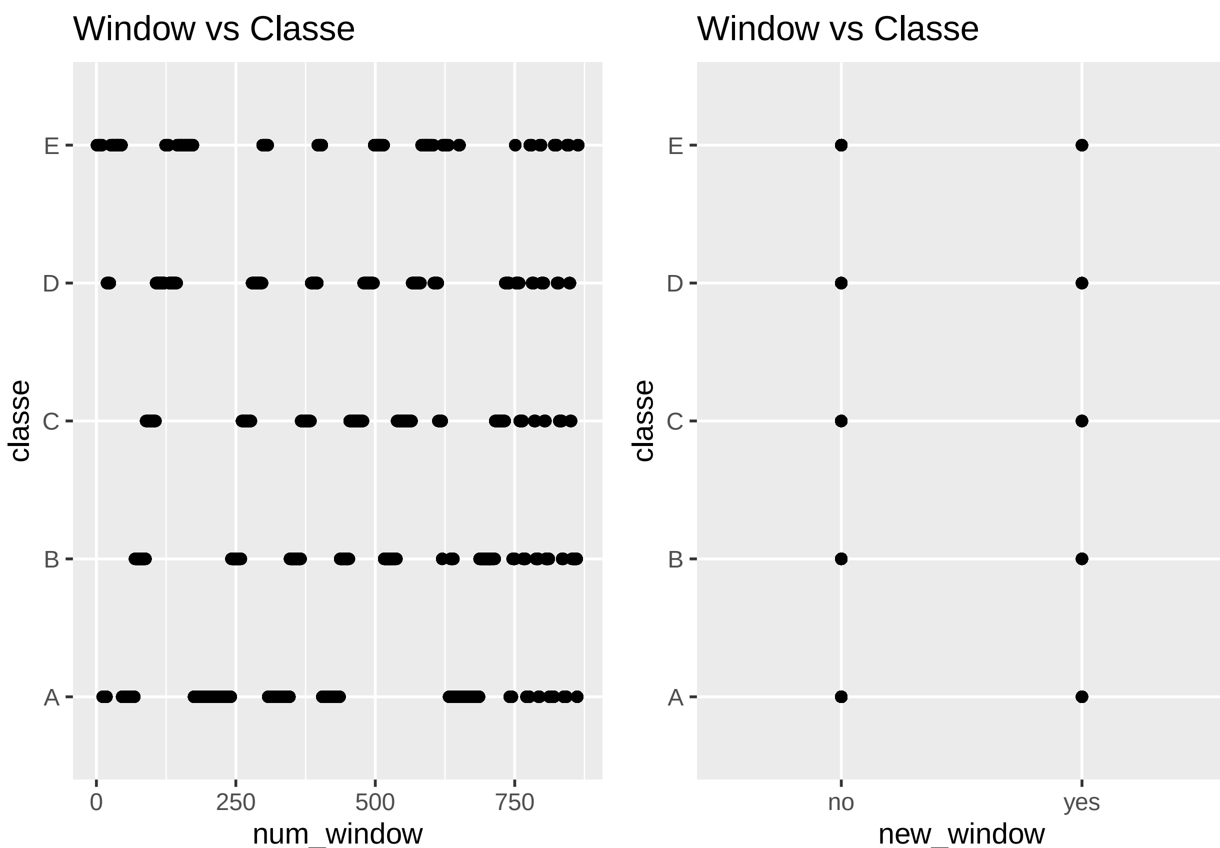
```
training <- training %>%
  select_if(~ !any(is.na(.)))
```

5. Looking for correlations between time and classe

```
p1 <- ggplot(training, aes(x = raw_timestamp_part_1, y = classe)) + geom_point() + ggtitle("Timestamp vs
p2 <- ggplot(training, aes(x = raw_timestamp_part_2, y = classe)) + geom_point() + ggtitle("Timestamp vs
grid.arrange(p1, p2, nrow = 1)
```



```r
p1 <- ggplot(training, aes(x = num_window, y = classe)) + geom_point() + ggtitle("Window vs Classe")
p2 <- ggplot(training, aes(x = new_window, y = classe)) + geom_point() + ggtitle("Window vs Classe")
grid.arrange(p1, p2, nrow = 1)
```



Since we couldn't detect a strong correlation between time and classe, we will exclude these variables from our prediction model

```r
training <- training[,-c(1,2,3,4,5,6,7)]
```

6. Training our models

First we will try to preprocess our data and then use the method rpart to build a classification tree

```r
preProc <- preProcess(training[, -53], method = c("center", "scale"))
trainTransformed <- predict(preProc, training)
testTransformed <- predict(preProc, testing)
modelFit <- train(classe ~ ., method = "rpart", data = trainTransformed)
modelFit
```

```
## CART
##
## 9812 samples
##   52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
```

```
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 9812, 9812, 9812, 9812, 9812, 9812, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.03703176  0.4818436  0.32291713
##   0.04250107  0.4419250  0.25739765
##   0.10938613  0.3208191  0.05989378
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.03703176.
```

```
A <- table(testTransformed$classe, predict(modelFit, newdata = testTransformed))
A
```

```
##
##        A    B    C    D    E
##   A 1727    6  748  303    5
##   B  296  337  675  600    0
##   C   40   39 1407  204    0
##   D   82   10  725  803    0
##   E   26   11  472  440  854
```

```
sum(diag(A))/sum(A)
```

```
## [1] 0.5227319
```

The accuracy is below 50% in the train and in the test sets, therefore this model isn't good enough for this dataset.

We will now fit a boosted tree model to see if accuracy can be improved.

```
modelFit <- train(classe ~ ., method = "gbm", data = training, verbose = FALSE)
modelFit
```

```
## Stochastic Gradient Boosting
##
## 9812 samples
##   52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 9812, 9812, 9812, 9812, 9812, 9812, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7522315  0.6861001
##   1                  100      0.8178687  0.7696358
##   1                  150      0.8486226  0.8085363
```

```
##   2                   50      0.8513406  0.8117788
##   2                  100      0.8990859  0.8723157
##   2                  150      0.9245012  0.9045044
##   3                   50      0.8903653  0.8612297
##   3                  100      0.9340653  0.9166054
##   3                  150      0.9531209  0.9407144
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
A <- table(testing$classe, predict(modelFit, newdata = testing))
A
```

```
##
##         A    B    C    D    E
##    A 2747   30    9    2    1
##    B   74 1764   64    5    1
##    C    0   46 1627   16    1
##    D    2    4   54 1551    9
##    E    5   19   14   16 1749
```

```
sum(diag(A))/sum(A)
```

```
## [1] 0.9620795
```

Processing time is considerably higher but it is a good trade off for the increase in accuracy, therefore we will use this option.

7. Final Prediction

```
predict(modelFit, final_test)
```

```
##  [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
```