Practical Machine Learning

Li Xiaowei

Summary

This report is a study on building a predication model to predict personal activity level. The input are the data collected from various wearable devices. The output is the classe range from “A” to “E”. More data and backgroup of the data from this study can be found from “http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)”.

Running environment of this report is as below.

```
sessionInfo()
```

```
## R version 3.2.1 (2015-06-18)
## Platform: x86_64-apple-darwin14.3.0 (64-bit)
## Running under: OS X 10.11 (unknown)
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] magrittr_1.5    formatR_1.2     tools_3.2.1     htmltools_0.2.6
##  [5] yaml_2.1.13     stringi_0.5-5   rmarkdown_0.7   knitr_1.10.5
##  [9] stringr_1.0.0   digest_0.6.8    evaluate_0.7
```

Download and prepare data

Prepare packages

This study is basically a machine learning of classification problem. There are quite some different learning algorithms and I choose random forest here.

```
if(require(randomForest) == FALSE) install.pacakge("randomForest")
library(randomForest)
if(require(caret) == FALSE) install.packages("caret")
library(caret)
```

Download and read data

```
#setwd("./Chapter8.Practical_Machine_Learning/Chapter8Project/")
download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",
              "pml_training.csv", method = "curl")
download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",
              "pml_testing.csv", method = "curl")

# The "pml_training.csv" file contains all the training and validating data
pmlTraining <- read.csv("pml_training.csv")
# "pml_testing.csv" is for the purpose of this project 20 testing cases
testing <- read.csv("pml_testing.csv")
```

The *testing.csv* file contains only the 20 testing cases. The *pml_testing.csv* file contains the training and validating data. Here I split it at 7/3. 70% for training and 30% for validating to evaluate the trained model.

```
# Split data to training and validating with 7/3
inTrain <- createDataPartition(y = pmlTraining$classe, p = 0.7, list = FALSE)
training <- pmlTraining[inTrain, ]
validating <- pmlTraining[-inTrain, ]
```

Le’t take a glance at the data first.

```
dim(training)
```

```
## [1] 13737   160
```

```
str(testing)
```

```
## 'data.frame':    20 obs. of  160 variables:
##  $ X                       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ user_name               : Factor w/ 6 levels "adelmo","carlitos",..: 6 5 5 1 4 5 5 5 2 3 ...
##  $ raw_timestamp_part_1    : int  1323095002 1322673067 1322673075 1322832789 1322489635 1322673149 1322673128
1322673076 1323084240 1322837822 ...
##  $ raw_timestamp_part_2    : int  868349 778725 342967 560311 814776 510661 766645 54671 916313 384285 ...
##  $ cvtd_timestamp          : Factor w/ 11 levels "02/12/2011 13:33",..: 5 10 10 1 6 11 11 10 3 2 ...
##  $ new_window              : Factor w/ 1 level "no": 1 1 1 1 1 1 1 1 1 1 ...
##  $ num_window              : int  74 431 439 194 235 504 485 440 323 664 ...
##  $ roll_belt               : num  123 1.02 0.87 125 1.35 -5.92 1.2 0.43 0.93 114 ...
##  $ pitch_belt              : num  27 4.87 1.82 -41.6 3.33 1.59 4.44 4.15 6.72 22.4 ...
##  $ yaw_belt                : num  -4.75 -88.9 -88.5 162 -88.6 -87.7 -87.3 -88.5 -93.7 -13.1 ...
##  $ total_accel_belt        : int  20 4 5 17 3 4 4 4 4 18 ...
##  $ kurtosis_roll_belt      : logi  NA NA NA NA NA NA ...
##  $ kurtosis_picth_belt     : logi  NA NA NA NA NA NA ...
##  $ kurtosis_yaw_belt       : logi  NA NA NA NA NA NA ...
##  $ skewness_roll_belt      : logi  NA NA NA NA NA NA ...
##  $ skewness_roll_belt.1    : logi  NA NA NA NA NA NA ...
##  $ skewness_yaw_belt       : logi  NA NA NA NA NA NA ...
##  $ max_roll_belt           : logi  NA NA NA NA NA NA ...
##  $ max_picth_belt          : logi  NA NA NA NA NA NA ...
##  $ max_yaw_belt            : logi  NA NA NA NA NA NA ...
##  $ min_roll_belt           : logi  NA NA NA NA NA NA ...
##  $ min_pitch_belt          : logi  NA NA NA NA NA NA ...
##  $ min_yaw_belt            : logi  NA NA NA NA NA NA ...
##  $ amplitude_roll_belt     : logi  NA NA NA NA NA NA ...
##  $ amplitude_pitch_belt    : logi  NA NA NA NA NA NA ...
##  $ amplitude_yaw_belt      : logi  NA NA NA NA NA NA ...
##  $ var_total_accel_belt    : logi  NA NA NA NA NA NA ...
##  $ avg_roll_belt           : logi  NA NA NA NA NA NA ...
##  $ stddev_roll_belt        : logi  NA NA NA NA NA NA ...
##  $ var_roll_belt           : logi  NA NA NA NA NA NA ...
##  $ avg_pitch_belt          : logi  NA NA NA NA NA NA ...
##  $ stddev_pitch_belt       : logi  NA NA NA NA NA NA ...
##  $ var_pitch_belt          : logi  NA NA NA NA NA NA ...
##  $ avg_yaw_belt            : logi  NA NA NA NA NA NA ...
##  $ stddev_yaw_belt         : logi  NA NA NA NA NA NA ...
##  $ var_yaw_belt            : logi  NA NA NA NA NA NA ...
##  $ gyros_belt_x            : num  -0.5 -0.06 0.05 0.11 0.03 0.1 -0.06 -0.18 0.1 0.14 ...
##  $ gyros_belt_y            : num  -0.02 -0.02 0.02 0.11 0.02 0.05 0 -0.02 0 0.11 ...
##  $ gyros_belt_z            : num  -0.46 -0.07 0.03 -0.16 0 -0.13 0 -0.03 -0.02 -0.16 ...
##  $ accel_belt_x            : int  -38 -13 1 46 -8 -11 -14 -10 -15 -25 ...
##  $ accel_belt_y            : int  69 11 -1 45 4 -16 2 -2 1 63 ...
##  $ accel_belt_z            : int  -179 39 49 -156 27 38 35 42 32 -158 ...
##  $ magnet_belt_x           : int  -13 43 29 169 33 31 50 39 -6 10 ...
##  $ magnet_belt_y           : int  581 636 631 608 566 638 622 635 600 601 ...
##  $ magnet_belt_z           : int  -382 -309 -312 -304 -418 -291 -315 -305 -302 -330 ...
##  $ roll_arm                : num  40.7 0 0 -109 76.1 0 0 0 -137 -82.4 ...
##  $ pitch_arm               : num  -27.8 0 0 55 2.76 0 0 0 11.2 -63.8 ...
##  $ yaw_arm                 : num  178 0 0 -142 102 0 0 0 -167 -75.3 ...
##  $ total_accel_arm         : int  10 38 44 25 29 14 15 22 34 32 ...
##  $ var_accel_arm           : logi  NA NA NA NA NA NA ...
##  $ avg_roll_arm            : logi  NA NA NA NA NA NA ...
##  $ stddev_roll_arm         : logi  NA NA NA NA NA NA ...
##  $ var_roll_arm            : logi  NA NA NA NA NA NA ...
##  $ avg_pitch_arm           : logi  NA NA NA NA NA NA ...
##  $ stddev_pitch_arm        : logi  NA NA NA NA NA NA ...
##  $ var_pitch_arm           : logi  NA NA NA NA NA NA ...
##  $ avg_yaw_arm             : logi  NA NA NA NA NA NA ...
##  $ stddev_yaw_arm          : logi  NA NA NA NA NA NA ...
##  $ var_yaw_arm             : logi  NA NA NA NA NA NA ...
##  $ gyros_arm_x             : num  -1.65 -1.17 2.1 0.22 -1.96 0.02 2.36 -3.71 0.03 0.26 ...
##  $ gyros_arm_y             : num  0.48 0.85 -1.36 -0.51 0.79 0.05 -1.01 1.85 -0.02 -0.5 ...
##  $ gyros_arm_z             : num  -0.18 -0.43 1.13 0.92 -0.54 -0.07 0.89 -0.69 -0.02 0.79 ...
##  $ accel_arm_x             : int  16 -290 -341 -238 -197 -26 99 -98 -287 -301 ...
##  $ accel_arm_y             : int  38 215 245 -57 200 130 79 175 111 -42 ...
##  $ accel_arm_z             : int  93 -90 -87 6 -30 -19 -67 -78 -122 -80 ...
##  $ magnet_arm_x            : int  -326 -325 -264 -173 -170 396 702 535 -367 -420 ...
##  $ magnet_arm_y            : int  385 447 474 257 275 176 15 215 335 294 ...
##  $ magnet_arm_z            : int  481 434 413 633 617 516 217 385 520 493 ...
##  $ kurtosis_roll_arm       : logi  NA NA NA NA NA NA ...
##  $ kurtosis_picth_arm      : logi  NA NA NA NA NA NA ...
##  $ kurtosis_yaw_arm        : logi  NA NA NA NA NA NA ...
##  $ skewness_roll_arm       : logi  NA NA NA NA NA NA ...
##  $ skewness_pitch_arm      : logi  NA NA NA NA NA NA ...
##  $ skewness_yaw_arm        : logi  NA NA NA NA NA NA ...
##  $ max_roll_arm            : logi  NA NA NA NA NA NA ...
##  $ max_picth_arm           : logi  NA NA NA NA NA NA ...
##  $ max_yaw_arm             : logi  NA NA NA NA NA NA ...
##  $ min_roll_arm            : logi  NA NA NA NA NA NA ...
##  $ min_pitch_arm           : logi  NA NA NA NA NA NA ...
##  $ min_yaw_arm             : logi  NA NA NA NA NA NA ...
##  $ amplitude_roll_arm      : logi  NA NA NA NA NA NA ...
```

```
##  $ amplitude_pitch_arm     : logi  NA NA NA NA NA NA ...
##  $ amplitude_yaw_arm       : logi  NA NA NA NA NA NA ...
##  $ roll_dumbbell           : num  -17.7 54.5 57.1 43.1 -101.4 ...
##  $ pitch_dumbbell          : num  25 -53.7 -51.4 -30 -53.4 ...
##  $ yaw_dumbbell            : num  126.2 -75.5 -75.2 -103.3 -14.2 ...
##  $ kurtosis_roll_dumbbell  : logi  NA NA NA NA NA NA ...
##  $ kurtosis_picth_dumbbell : logi  NA NA NA NA NA NA ...
##  $ kurtosis_yaw_dumbbell   : logi  NA NA NA NA NA NA ...
##  $ skewness_roll_dumbbell  : logi  NA NA NA NA NA NA ...
##  $ skewness_pitch_dumbbell : logi  NA NA NA NA NA NA ...
##  $ skewness_yaw_dumbbell   : logi  NA NA NA NA NA NA ...
##  $ max_roll_dumbbell       : logi  NA NA NA NA NA NA ...
##  $ max_picth_dumbbell      : logi  NA NA NA NA NA NA ...
##  $ max_yaw_dumbbell        : logi  NA NA NA NA NA NA ...
##  $ min_roll_dumbbell       : logi  NA NA NA NA NA NA ...
##  $ min_pitch_dumbbell      : logi  NA NA NA NA NA NA ...
##  $ min_yaw_dumbbell        : logi  NA NA NA NA NA NA ...
##  $ amplitude_roll_dumbbell : logi  NA NA NA NA NA NA ...
##   [list output truncated]
```

```
summary(testing)
```

The original data set has 160 columns. Before fitting a model, we need to explore some data cleaning. The structure of testing data set gives really a lot of hints on how to clean the data. And I found just by eliminating the columns with NAs will produce very clean data set. Blow code shows the steps.

```
# Clean data by elimitating logical columns according to the testing data
t <- list()
for(i in 6:159) {
      if(class(testing[,i]) == "logical") t[length(t) + 1] <- i
}

newTraining <- training[, -c(1:5, unlist(t))]
newValidating <- validating[, -c(1:5, unlist(t))]
newTesting <- testing[, -c(1:5, unlist(t))]
```

The resulted data sets have columns of either integer or numeric data type. To make it easier for the remaining of this report, I coerced interger column to numeric. This is very helpful when come to the stage of predicting the testing data set.

```
# Converting data type as all columns are either integer or numeric
colNum <- dim(newTesting)[2] - 1
for(i in 1:colNum) {
        newTraining[, i] <- as.numeric(newTraining[, i])
        newValidating[, i] <- as.numeric(newValidating[, i])
        newTesting[, i] <- as.numeric(newTesting[, i])
}
```

Fit in the training model

I started the first random forest model with *mtry = 7* and *ntree = 100*. On my macbook it takes about one minute to complete. And the fine tuning will focus on these two parameters.

```
# It takes about one minute to finish the training with mtry = 7 and ntree = 100
startTime <- proc.time()
rfModFit <- randomForest(classe ~ ., data = newTraining,
                         mtry = floor(sqrt(dim(newTraining)[2] - 1)),
                         ntree = 100,
                         proximity = TRUE,
                         importance = TRUE)
proc.time() - startTime
```

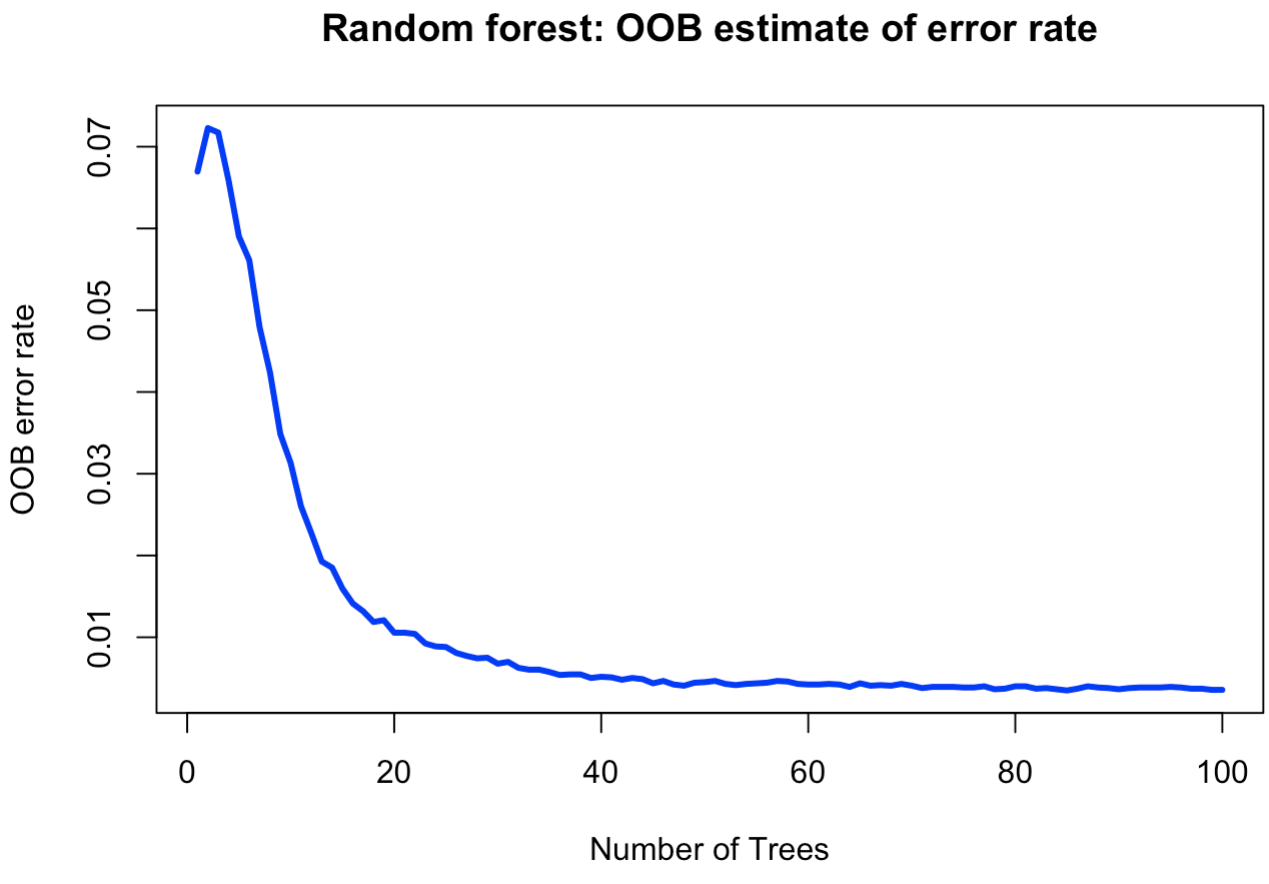
```
##    user  system elapsed
##  64.050   1.863  66.166
```

Fine tune the model

Parameter ntree

Below plot shows the relationship of OOB error rate versus the number of trees. It can be observed that, started from 40, OOB wouldn't drop much. This observation will be used to fine tune the final training model.

```
plot(rfModFit$err.rate[, 1], type = "l", lwd = 3, col = "blue",
     main = "Random forest: OOB estimate of error rate",
     xlab = "Number of Trees", ylab = "OOB error rate")
```



Parameter mtry

Another parameter to minimize the OOB error is the *mtry*, below piece of code and plot shows how.

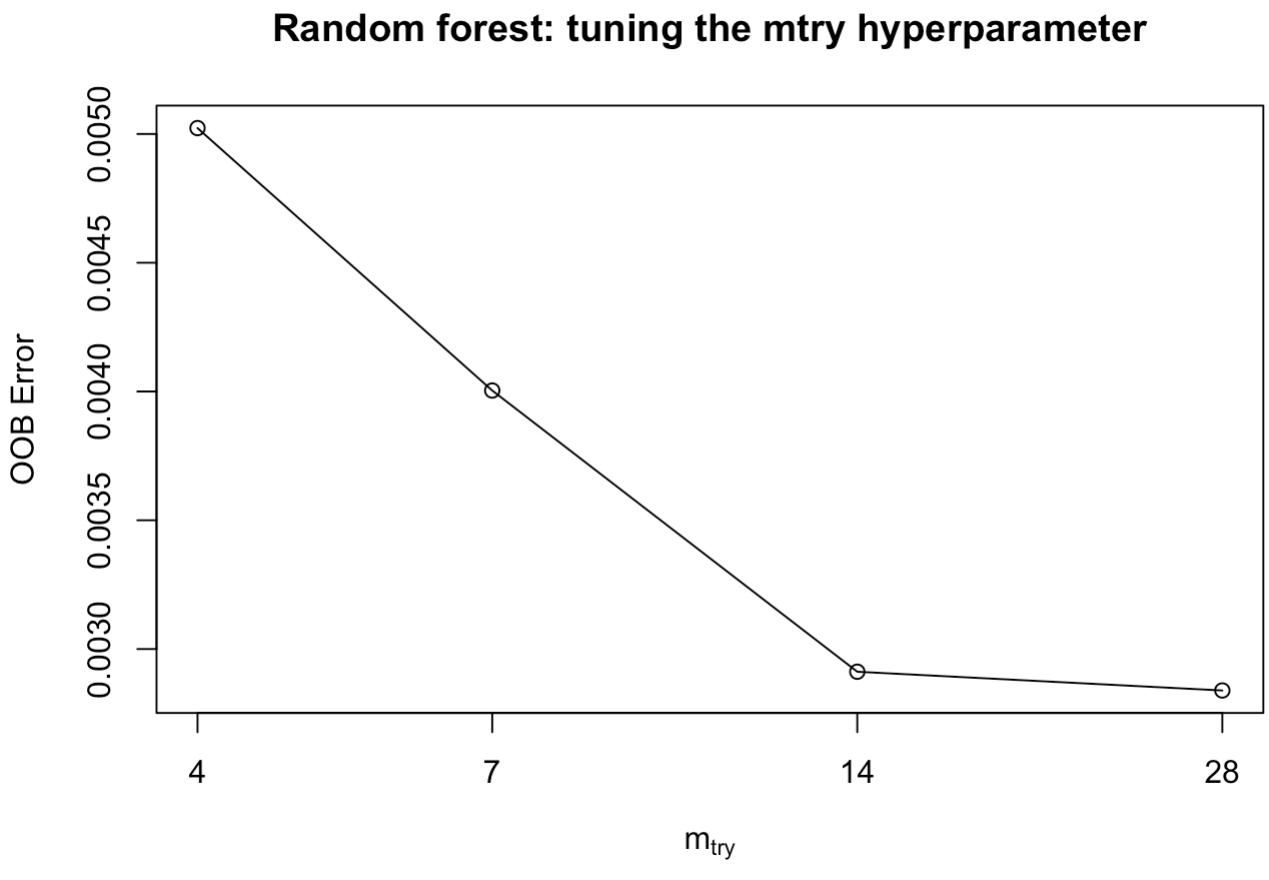
mtry of 14 to 28 will produce the smaller OOB error and I'll choose 14.

```
tuneRF(subset(newTraining, select = -classe),
       newTraining$classe,
       ntreeTry = 100)
```

```
## mtry = 7  OOB error = 0.4%
## Searching left ...
## mtry = 4     OOB error = 0.5%
## -0.2545455 0.05
## Searching right ...
## mtry = 14    OOB error = 0.29%
## 0.2727273 0.05
## mtry = 28    OOB error = 0.28%
## 0.025 0.05
```

```
##        mtry    OOBError
## 4.OOB     4 0.005022931
## 7.OOB     7 0.004003785
## 14.OOB   14 0.002911844
## 28.OOB   28 0.002839048
```

```
title("Random forest: tuning the mtry hyperparameter")
```

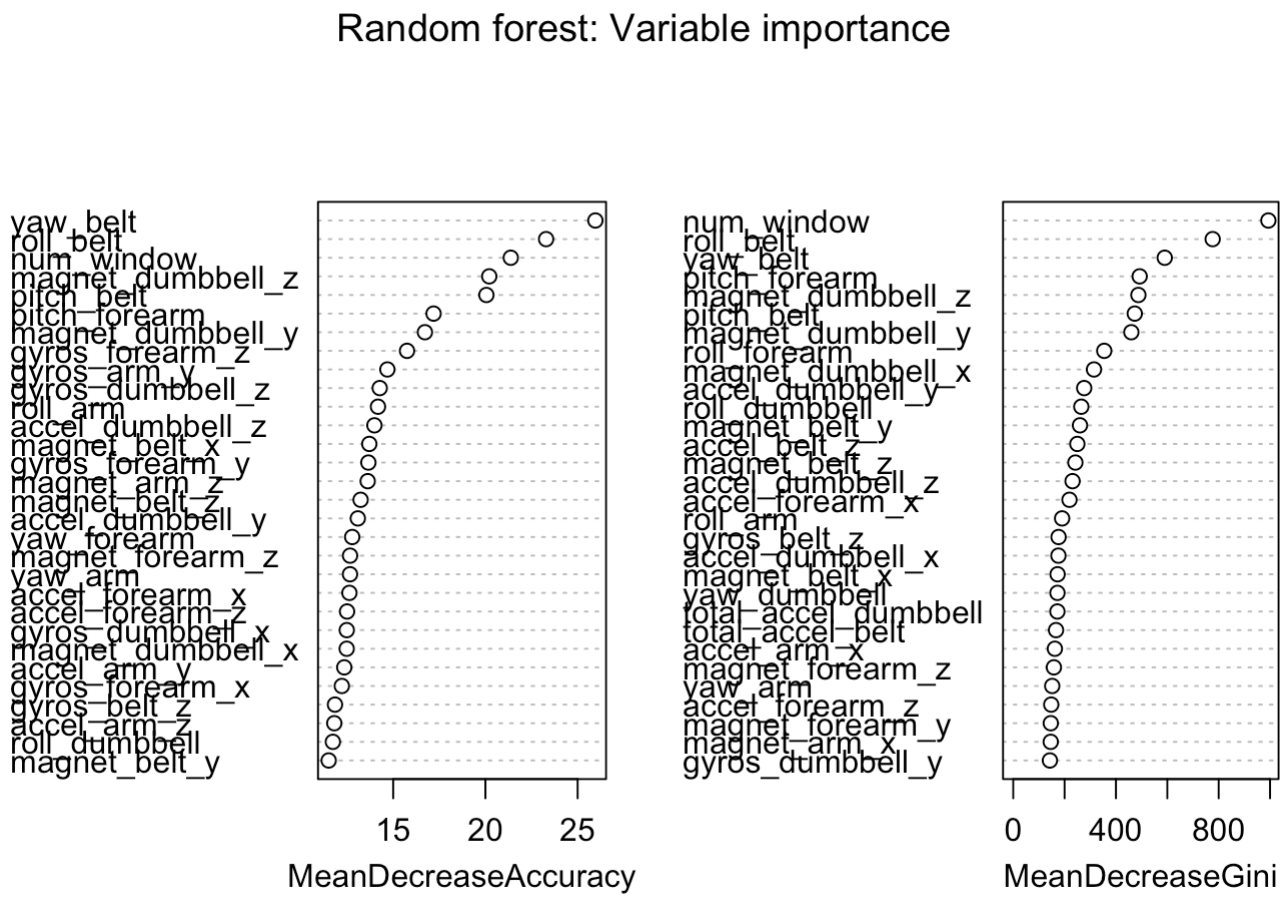


The OOB error is very small from previous two analysis, it's smaller than 1%. We would expect the out of sample error also very small if we fit the validating data set to the model. We'll see this in the coming sections.

Variable importance

There are 54 predictors in the data set. Some of them can be excluded without causing much decline in accuracy. There is a very nice function can do this, refer to below plot.

```
varImpPlot(rfModFit, main = "Random forest: Variable importance")
```



Validating the model before fine tuning

Now let's feed in the validating data set to check the performance of the model.

```
rfPred <- predict(rfModFit, subset(newValidating, select = -classe))
table(rfPred, newValidating$classe)
```

```
##
## rfPred    A    B    C    D    E
##      A 1674    0    0    0    0
##      B    0 1139    2    0    0
##      C    0    0 1024    4    0
##      D    0    0    0  959    0
##      E    0    0    0    1 1082
```

```
sum(diag(table(rfPred, newValidating$classe)))/nrow(newValidating)
```

```
## [1] 0.9988105
```

The out of sample error rate is 0.0011895.

Validating the model after fine tuning

The variable importance plot was shown previously, due to the limitation of the report length, I won't explore on reducing the predictors here. I'll update *ntree* to 40 and *mtry* to 14 according to the previous fine tuning section. Then compare the training time and performance with the first model.

```
startTime <- proc.time()
rfModFit2 <- randomForest(classe ~ ., data = newTraining,
                          mtry = 14,
                          ntree = 40,
                          proximity = TRUE,
                          importance = TRUE)
proc.time() - startTime
```

```
##    user  system elapsed
##  32.162   1.782  34.185
```

With reduced *ntree* value, the training time was cut by half, only half a minute now. The analysis of impact of different *mtry* and *ntree* is omitted as it simply repeats the previous fine tune section.

Out of sample error rate is caculated by fit in the testing data set.

```
rfPred <- predict(rfModFit2, subset(newValidating, select = -classe))
table(rfPred, newValidating$classe)
```

```
##
## rfPred    A    B    C    D    E
##      A 1673    0    0    0    0
##      B    1 1138    0    0    0
##      C    0    1 1026    4    0
##      D    0    0    0  959    0
##      E    0    0    0    1 1082
```

```
sum(diag(table(rfPred, newValidating$classe)))/nrow(newValidating)
```

```
## [1] 0.9988105
```

After fine tune, the error rate is 0.0011895, which is almost the same as the first try (sometimes even exactly the same). But we achieved half training time. Moreover, as expected previously, this error rate is very small.

More other training algorithms also worth to try, like *gbm*, *bagging* and etcs. However, not covered in this report.

Answers to the 20 testing cases

This section will utilize the sample code provided to generate the answers of the 20 testing case. Passing rate 20/20.

```
# Save the model
saveRDS(rfModFit2, file="myFile.rds")

# Load the model
#rfModFit2 = readRDS("myFile.rds")

# Predication on testing data
answers <- predict(rfModFit2, subset(newTesting, select = - problem_id))
pml_write_files = function(x){
        n = length(x)
        for(i in 1:n){
                filename = paste0("problem_id_",i,".txt")
                write.table(x[i],file=filename,quote=FALSE,row.names=FALSE,col.names=FALSE)
        }
}
pml_write_files(answers)
```