Coursera - Practical Machine Learning Project

Kelvin Leung

7/22/2018

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://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Load libraries

```
library(caret)
```

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

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

```
library(rattle)
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.1.0 Copyright (c) 2006-2017 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(party)
```

```
## Loading required package: grid
```

```
## Loading required package: mvtnorm
```

```
## Loading required package: modeltools
```

```
## Loading required package: stats4
```

```
## Loading required package: strucchange
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: sandwich
```

```
library(rpart)
library(rpart.plot)
```

```r
library(RColorBrewer)
library(cvTools)
```

```
## Loading required package: robustbase
```

```r
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:rattle':
##
##     importance
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

Data loading

```r
train_data <- read.csv(url("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"), head
dim(train_data)
```

```
## [1] 19622   160
```

```r
test_data <- read.csv(url("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"), heade
dim(test_data)
```

```
## [1]  20 160
```

The training set has 19622 observations and each observation has 160 columns. We notice that many columns have N/A values or blank values. So we will remove them because they will not produce any information. Also, the first seven columns give information about the people who did the test and the timestamps. We will remove these columns in our model. Note, the "classe" variable is in the last column of our training set.

The testing set has 20 cases. It will be used to test the accuracy of our models.

Cleansing procedure

Here is the R code to remove the columns that has N/A or "" values.

```r
# removing columns having value of "N/A" or "" value that have at least 90% of the total number of rows
tidx_2remove <- which(colSums(is.na(train_data) | train_data=='') > 0.9* dim(train_data)[1])
# removing those columns
train_clean <- train_data[ ,-tidx_2remove]
# removing the first 7 columns that are irrelevant to the prediction model
train_clean <- train_clean[ ,-(1:7)]
dim(train_clean)
```

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

```r
str(train_clean)
```

```
## 'data.frame':    19622 obs. of  53 variables:
##  $ roll_belt           : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45 ...
##  $ pitch_belt          : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 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 ...
##  $ gyros_belt_x        : num  0 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 0.03 ...
##  $ gyros_belt_y        : num  0 0 0 0 0.02 0 0 0 0 0 ...
##  $ gyros_belt_z        : num  -0.02 -0.02 -0.02 -0.03 -0.02 -0.02 -0.02 -0.02 -0.02 0 ...
##  $ accel_belt_x        : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21 ...
##  $ accel_belt_y        : int  4 4 5 3 2 4 3 4 2 4 ...
##  $ accel_belt_z        : int  22 22 23 21 24 21 21 21 24 22 ...
##  $ magnet_belt_x       : int  -3 -7 -2 -6 -6 0 -4 -2 1 -3 ...
##  $ magnet_belt_y       : int  599 608 600 604 600 603 599 603 602 609 ...
##  $ magnet_belt_z       : int  -313 -311 -305 -310 -302 -312 -311 -313 -312 -308 ...
##  $ roll_arm            : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128 ...
##  $ pitch_arm           : num  22.5 22.5 22.5 22.1 22.1 22 21.9 21.8 21.7 21.6 ...
##  $ yaw_arm             : num  -161 -161 -161 -161 -161 -161 -161 -161 -161 -161 ...
##  $ total_accel_arm     : int  34 34 34 34 34 34 34 34 34 34 ...
##  $ gyros_arm_x         : num  0 0.02 0.02 0.02 0 0.02 0 0.02 0.02 0.02 ...
##  $ gyros_arm_y         : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03 ...
##  $ gyros_arm_z         : num  -0.02 -0.02 -0.02 0.02 0 0 0 0 -0.02 -0.02 ...
##  $ accel_arm_x         : int  -288 -290 -289 -289 -289 -289 -289 -289 -288 -288 ...
##  $ accel_arm_y         : int  109 110 110 111 111 111 111 111 109 110 ...
##  $ accel_arm_z         : int  -123 -125 -126 -123 -123 -122 -125 -124 -122 -124 ...
##  $ magnet_arm_x        : int  -368 -369 -368 -372 -374 -369 -373 -372 -369 -376 ...
##  $ magnet_arm_y        : int  337 337 344 344 337 342 336 338 341 334 ...
##  $ magnet_arm_z        : int  516 513 513 512 506 513 509 510 518 516 ...
##  $ roll_dumbbell       : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell      : num  -70.5 -70.6 -70.3 -70.4 -70.4 ...
##  $ yaw_dumbbell        : num  -84.9 -84.7 -85.1 -84.9 -84.9 ...
##  $ total_accel_dumbbell: int  37 37 37 37 37 37 37 37 37 37 ...
##  $ gyros_dumbbell_x    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ gyros_dumbbell_y    : num  -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 ...
##  $ gyros_dumbbell_z    : num  0 0 0 -0.02 0 0 0 0 0 0 ...
##  $ accel_dumbbell_x    : int  -234 -233 -232 -232 -233 -234 -232 -234 -232 -235 ...
##  $ accel_dumbbell_y    : int  47 47 46 48 48 48 47 46 47 48 ...
##  $ accel_dumbbell_z    : int  -271 -269 -270 -269 -270 -269 -270 -272 -269 -270 ...
##  $ magnet_dumbbell_x   : int  -559 -555 -561 -552 -554 -558 -551 -555 -549 -558 ...
##  $ magnet_dumbbell_y   : int  293 296 298 303 292 294 295 300 292 291 ...
##  $ magnet_dumbbell_z   : num  -65 -64 -63 -60 -68 -66 -70 -74 -65 -69 ...
##  $ roll_forearm        : num  28.4 28.3 28.3 28.1 28 27.9 27.9 27.8 27.7 27.7 ...
##  $ pitch_forearm       : num  -63.9 -63.9 -63.9 -63.9 -63.9 -63.9 -63.9 -63.8 -63.8 -63.8 ...
##  $ yaw_forearm         : num  -153 -153 -152 -152 -152 -152 -152 -152 -152 -152 ...
##  $ total_accel_forearm : int  36 36 36 36 36 36 36 36 36 36 ...
##  $ gyros_forearm_x     : num  0.03 0.02 0.03 0.02 0.02 0.02 0.02 0.02 0.03 0.02 ...
##  $ gyros_forearm_y     : num  0 0 -0.02 -0.02 0 -0.02 0 -0.02 0 0 ...
##  $ gyros_forearm_z     : num  -0.02 -0.02 0 0 -0.02 -0.03 -0.02 0 -0.02 -0.02 ...
##  $ accel_forearm_x     : int  192 192 196 189 189 193 195 193 193 190 ...
##  $ accel_forearm_y     : int  203 203 204 206 206 203 205 205 204 205 ...
##  $ accel_forearm_z     : int  -215 -216 -213 -214 -214 -215 -215 -213 -214 -215 ...
##  $ magnet_forearm_x    : int  -17 -18 -18 -16 -17 -9 -18 -9 -16 -22 ...
##  $ magnet_forearm_y    : num  654 661 658 658 655 660 659 660 653 656 ...
##  $ magnet_forearm_z    : num  476 473 469 469 473 478 470 474 476 473 ...
##  $ classe              : Factor w/ 5 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```r
# removing columns having value of "N/A" or "" value that have at least 90% of the total number of rows
tidx_2remove <- which(colSums(is.na(test_data) | test_data=='') > 0.9* dim(test_data)[1])
# removing those columns
test_clean <- test_data[ ,-tidx_2remove]
# removing the first 7 columns that are irrelevant to the prediction model
test_clean <- test_clean[,-(1:7)]
dim(test_clean)
```

```
## [1] 20 53
```

```r
str(test_clean)
```

```
## 'data.frame':    20 obs. of  53 variables:
##  $ 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 ...
##  $ 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 ...
##  $ 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 ...
##  $ 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 ...
##  $ total_accel_dumbbell: int  9 31 29 18 4 29 29 29 3 2 ...
##  $ gyros_dumbbell_x    : num  0.64 0.34 0.39 0.1 0.29 -0.59 0.34 0.37 0.03 0.42 ...
##  $ gyros_dumbbell_y    : num  0.06 0.05 0.14 -0.02 -0.47 0.8 0.16 0.14 -0.21 0.51 ...
##  $ gyros_dumbbell_z    : num  -0.61 -0.71 -0.34 0.05 -0.46 1.1 -0.23 -0.39 -0.21 -0.03 ...
##  $ accel_dumbbell_x    : int  21 -153 -141 -51 -18 -138 -145 -140 0 -7 ...
##  $ accel_dumbbell_y    : int  -15 155 155 72 -30 166 150 159 25 -20 ...
##  $ accel_dumbbell_z    : int  81 -205 -196 -148 -5 -186 -190 -191 9 7 ...
##  $ magnet_dumbbell_x   : int  523 -502 -506 -576 -424 -543 -484 -515 -519 -531 ...
##  $ magnet_dumbbell_y   : int  -528 388 349 238 252 262 354 350 348 321 ...
##  $ magnet_dumbbell_z   : int  -56 -36 41 53 312 96 97 53 -32 -164 ...
##  $ roll_forearm        : num  141 109 131 0 -176 150 155 -161 15.5 13.2 ...
##  $ pitch_forearm       : num  49.3 -17.6 -32.6 0 -2.16 1.46 34.5 43.6 -63.5 19.4 ...
```

```
##  $ yaw_forearm         : num  156 106 93 0 -47.9 89.7 152 -89.5 -139 -105 ...
##  $ total_accel_forearm : int  33 39 34 43 24 43 32 47 36 24 ...
##  $ gyros_forearm_x     : num  0.74 1.12 0.18 1.38 -0.75 -0.88 -0.53 0.63 0.03 0.02 ...
##  $ gyros_forearm_y     : num  -3.34 -2.78 -0.79 0.69 3.1 4.26 1.8 -0.74 0.02 0.13 ...
##  $ gyros_forearm_z     : num  -0.59 -0.18 0.28 1.8 0.8 1.35 0.75 0.49 -0.02 -0.07 ...
##  $ accel_forearm_x     : int  -110 212 154 -92 131 230 -192 -151 195 -212 ...
##  $ accel_forearm_y     : int  267 297 271 406 -93 322 170 -331 204 98 ...
##  $ accel_forearm_z     : int  -149 -118 -129 -39 172 -144 -175 -282 -217 -7 ...
##  $ magnet_forearm_x    : int  -714 -237 -51 -233 375 -300 -678 -109 0 -403 ...
##  $ magnet_forearm_y    : int  419 791 698 783 -787 800 284 -619 652 723 ...
##  $ magnet_forearm_z    : int  617 873 783 521 91 884 585 -32 469 512 ...
##  $ problem_id          : int  1 2 3 4 5 6 7 8 9 10 ...
```

From the above, the columns of "train_clean" match with the columns of "test_clean", except for the last column, "problem_id". But we do not need to care about the "problem_id" column at this time.

Build Classification models

We use "recursive partiaion tree (rpart)", "random forest (randomForest)", and "Stochastic Gradient Boosting (gbm)" to build classification models and compare their performance.

First, we partiion the training data into 2 parts and use cross validation method to validate the models we build.

To ensure the reproductivity of this experiment, we initial the seed to 12345.

```r
set.seed(12345)
dim(train_clean)
```

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

```r
dim(test_clean)
```

```
## [1] 20 53
```

```r
tr1 <- createDataPartition(train_clean$classe, p=0.6, list=FALSE)
training <- train_clean[tr1,]
testing <- train_clean[-tr1,]
```

Train with recursive partiaion tree (rpart)

```r
# copy from the manual page of trainControl
seeds <- vector(mode = "list", length = 51)
for(i in 1:50) seeds[[i]] <- sample.int(1000, 22)
seeds[[51]] <- sample.int(1000, 1)

# we set the seed to 12345 to generate rpart model with 2 fold validation
set.seed(12345)
ctrl2 <- trainControl(allowParallel=T, seeds = seeds, method="cv", number=2)
mDT2 <- train(classe~., data=training, method="rpart", model=TRUE, trControl=ctrl2)

# using the same seed to generate another rpart model with 10 fold validation
set.seed(12345)
ctrl10 <- trainControl(allowParallel=T, seeds= seeds, method="cv", number=10)
mDT10 <- train(classe~., data=training, method="rpart", model=TRUE, trControl=ctrl10)
```

Confusion matrix using out-of-sample data in the rpart tree model

We use "predict" to obtain out-of-sample predictions.

```
rtree_pred_x2 <- predict(mDT2, newdata=testing)
rtree_pred_x10 <- predict(mDT10, newdata=testing)

cm_rtree_pred_x2 <- confusionMatrix(rtree_pred_x2, testing$classe)
cm_rtree_pred_x10 <- confusionMatrix(rtree_pred_x10, testing$classe)
```

Showing the confusion matrix:

rpart model with 2-fold validation

```
cm_rtree_pred_x2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2224 1518 1368 1286  782
##          B    0    0    0    0    0
##          C    0    0    0    0    0
##          D    0    0    0    0    0
##          E    8    0    0    0  660
##
## Overall Statistics
##
##                Accuracy : 0.3676
##                  95% CI : (0.3569, 0.3784)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.1266
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9964   0.0000   0.0000   0.0000  0.45770
## Specificity            0.1176   1.0000   1.0000   1.0000  0.99875
## Pos Pred Value         0.3098      NaN      NaN      NaN  0.98802
## Neg Pred Value         0.9880   0.8065   0.8256   0.8361  0.89106
## Prevalence             0.2845   0.1935   0.1744   0.1639  0.18379
## Detection Rate         0.2835   0.0000   0.0000   0.0000  0.08412
## Detection Prevalence   0.9149   0.0000   0.0000   0.0000  0.08514
## Balanced Accuracy      0.5570   0.5000   0.5000   0.5000  0.72822
```

```
acc_cm_rtree_X2 <- round(cm_rtree_pred_x2$overall[1], 4)
```

rpart model with 10-fold validation

```
cm_rtree_pred_x10
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1357  229   38   66   15
##          B    3  259   28    8   10
##          C  693  725  819  389  557
##          D  171  305  483  823  200
##          E    8    0    0    0  660
##
## Overall Statistics
##
##                Accuracy : 0.4994
##                  95% CI : (0.4882, 0.5105)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3764
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.6080  0.17062   0.5987   0.6400  0.45770
## Specificity            0.9380  0.99226   0.6351   0.8233  0.99875
## Pos Pred Value         0.7959  0.84091   0.2573   0.4152  0.98802
## Neg Pred Value         0.8575  0.83298   0.8823   0.9210  0.89106
## Prevalence             0.2845  0.19347   0.1744   0.1639  0.18379
## Detection Rate         0.1730  0.03301   0.1044   0.1049  0.08412
## Detection Prevalence   0.2173  0.03926   0.4057   0.2526  0.08514
## Balanced Accuracy      0.7730  0.58144   0.6169   0.7316  0.72822
```

```
acc_cm_rtree_X10 <- round(cm_rtree_pred_x10$overall[1], 4)
```

Out-of-sample accuracy:

```
print.noquote(paste("Accurracy of rtree for CV n=2: ", acc_cm_rtree_X2))
```

```
## [1] Accurracy of rtree for CV n=2:  0.3676
```

```
print.noquote(paste("Accurracy of rtree for CV n=10: ", acc_cm_rtree_X10))
```

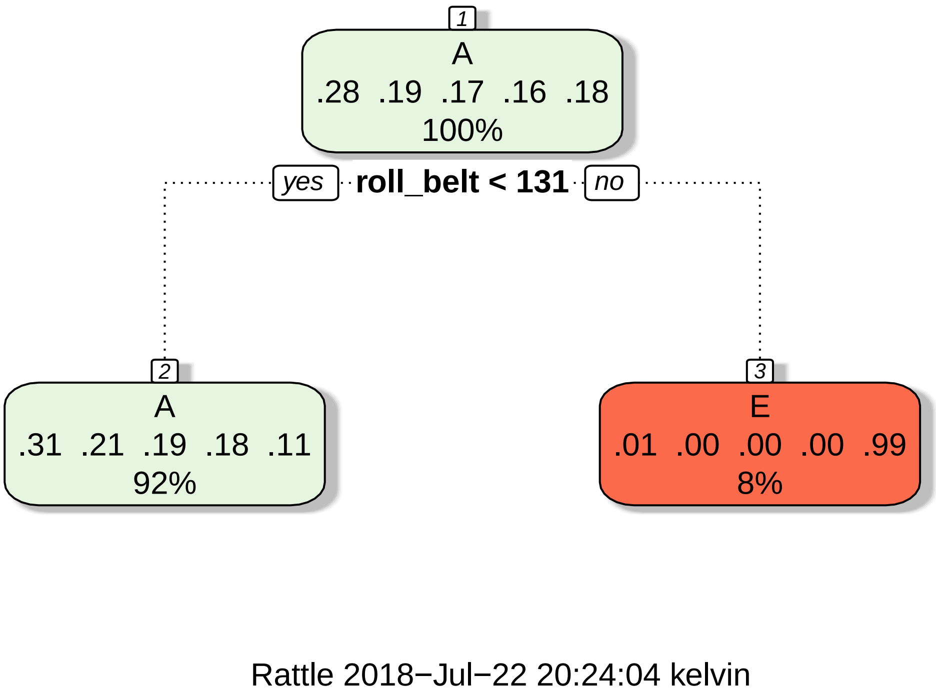
```
## [1] Accurracy of rtree for CV n=10:  0.4994
```

From the results shown above, the accuracy of rpart using cross-validation with n=10 is 0.4994 or 49.94%, which is more accurate than that with n=2 (i.e. 0.3676 or 36.76%).

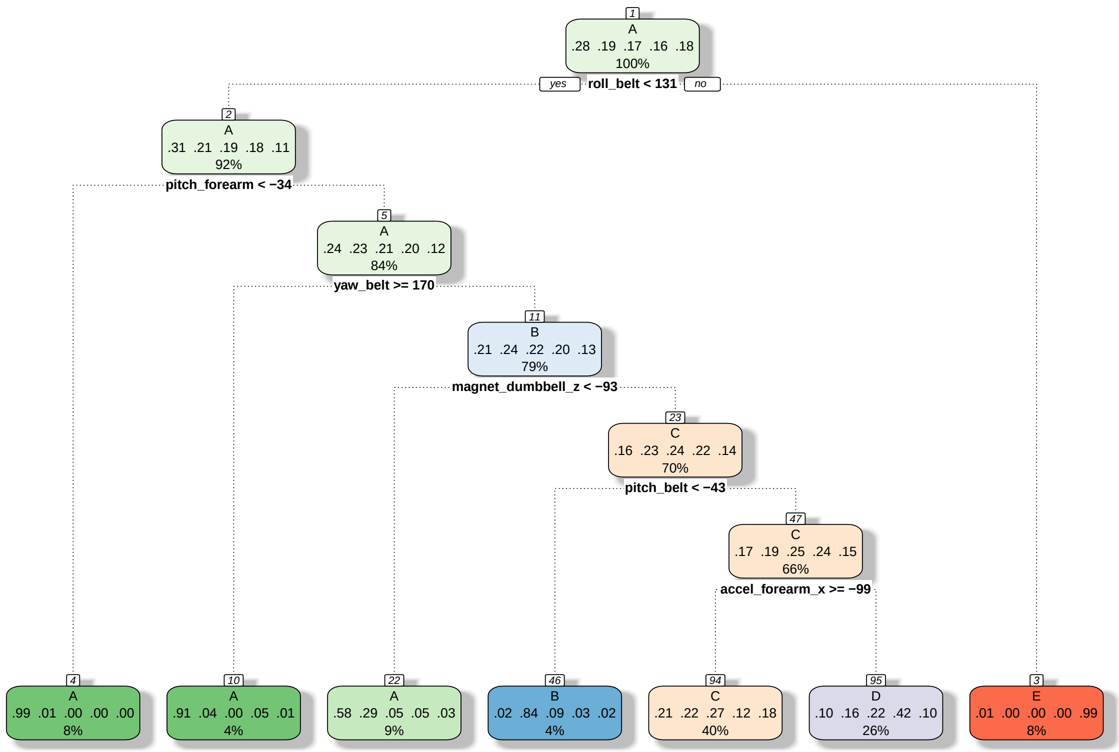
Also, the confusion matrix for rpart with 10-fold cross validation clearly shows that it has less confusion (i.e. spread away from a diagonal matrix or an identity matrix if we normalize the entires with the total number of entries) than that of rpart with 2-fold cross validation. Hence, 10-fold cross validation provides much better estimation performance than 2-fold cross validation model.

Plot of Decison Trees

```r
fancyRpartPlot(mDT2$finalModel)
```




```r
fancyRpartPlot(mDT10$finalModel)
```



Rattle 2018−Jul−22 20:24:05 kelvin

Random Forest

Training Random forest with 100 trees.

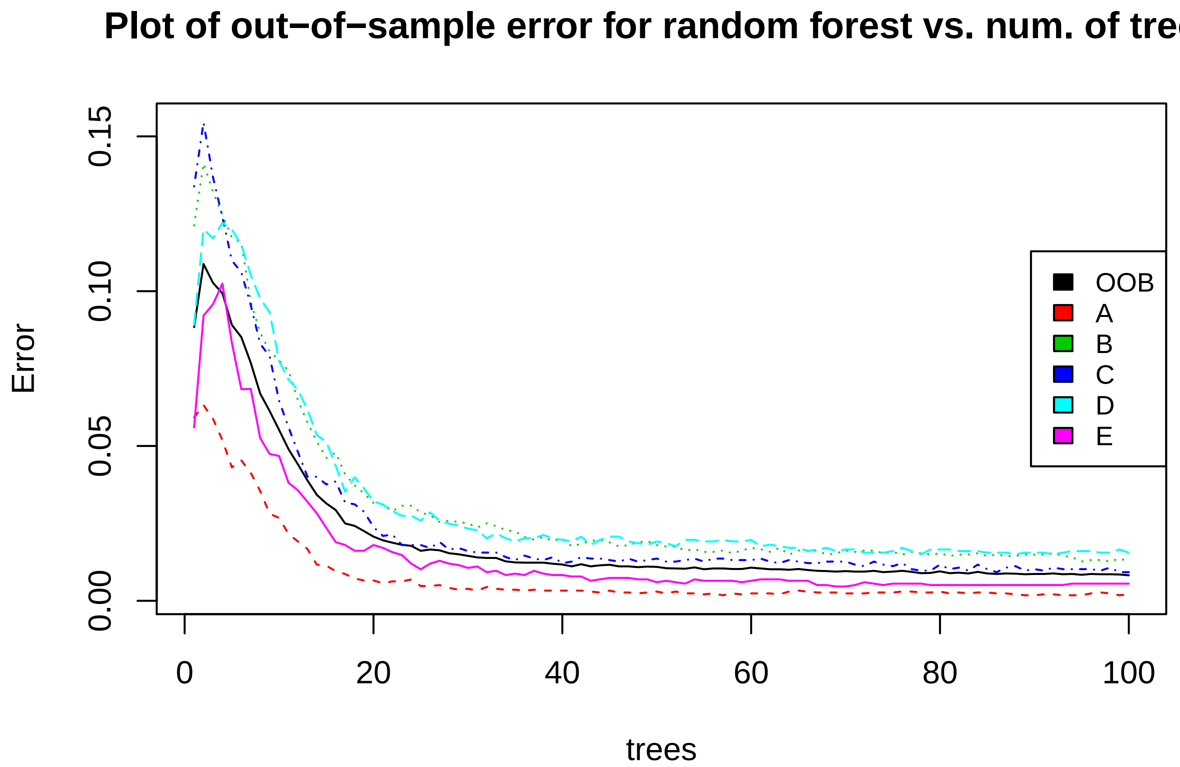
We generate a random forest model using 100 classifcation trees.

```
set.seed(12345)
mRF <- randomForest(classe ~., data=training, ntree=100, importance = TRUE)
```

I don't think it is neccessary to apply cross validation (or k-fold) in random forest becuase the performance of having out-of-bag in random forest is very simiar to cross validation. [see https://stats.stackexchange.com/questions/283760/is-cross-validation-unnecessary-for-random-forest].

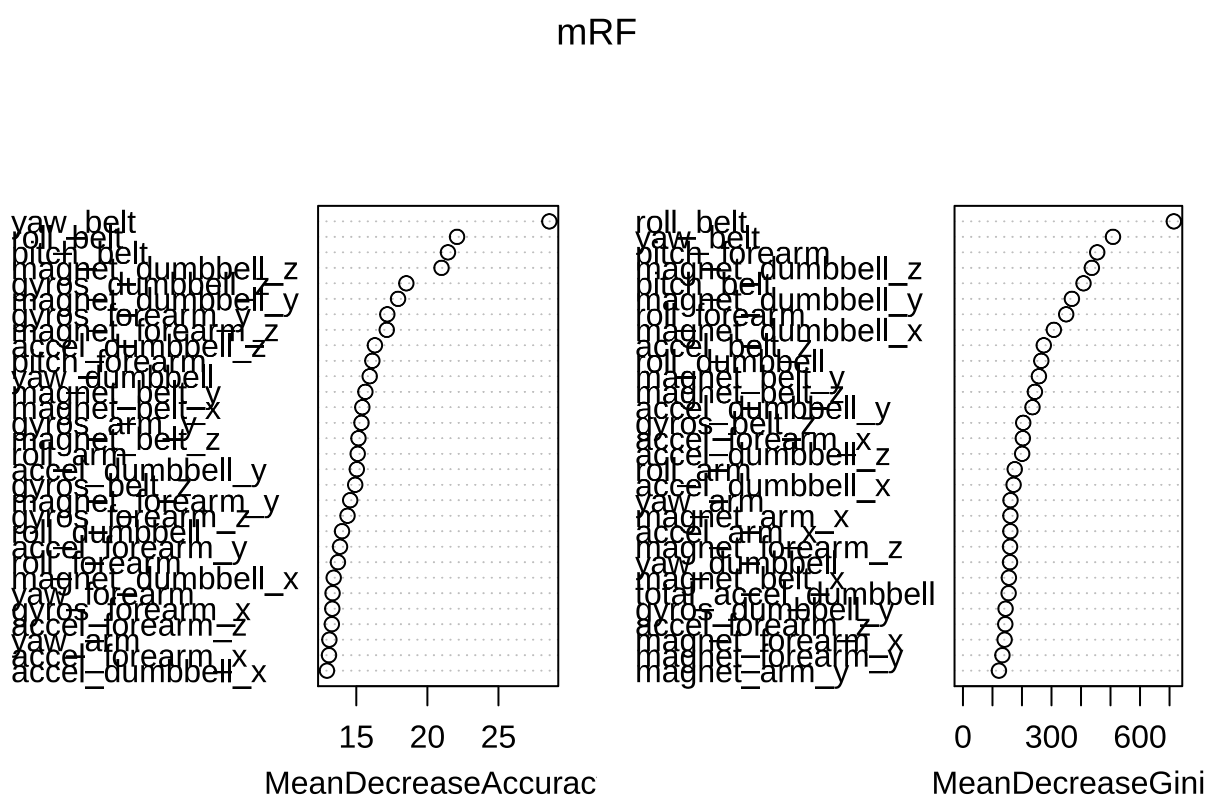
Plotting the out-of-sample error of the random forest vs. num. of trees

```
plot(mRF, main='Plot of out-of-sample error for random forest vs. num. of trees')
legend("right", colnames(mRF$err.rate), col=1:ncol(test_clean), cex=0.8, fill=1:ncol(test_data))
```



Plotting the important variables for the classification problem.

```
varImpPlot(mRF)
```



From the plots above, it shows that "mean descrease accuracy" and "mean decrease gini".

Confusion matrix using out-of-sample data in the random forest model

```
pred_rf <- predict(mRF, newdata=testing, type='class')
cm_rf <- confusionMatrix(pred_rf, testing$classe)
cm_rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2227    6    0    0    0
##          B    5 1506    4    0    0
##          C    0    6 1364   14    2
##          D    0    0    0 1269    4
##          E    0    0    0    3 1436
##
## Overall Statistics
##
##                Accuracy : 0.9944
##                  95% CI : (0.9925, 0.9959)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9929
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9978   0.9921   0.9971   0.9868   0.9958
## Specificity            0.9989   0.9986   0.9966   0.9994   0.9995
## Pos Pred Value         0.9973   0.9941   0.9841   0.9969   0.9979
## Neg Pred Value         0.9991   0.9981   0.9994   0.9974   0.9991
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2838   0.1919   0.1738   0.1617   0.1830
## Detection Prevalence   0.2846   0.1931   0.1767   0.1622   0.1834
## Balanced Accuracy      0.9983   0.9953   0.9968   0.9931   0.9977
```

Notice that the confusion matrix of random forest model is less confused than that of rpart model above.

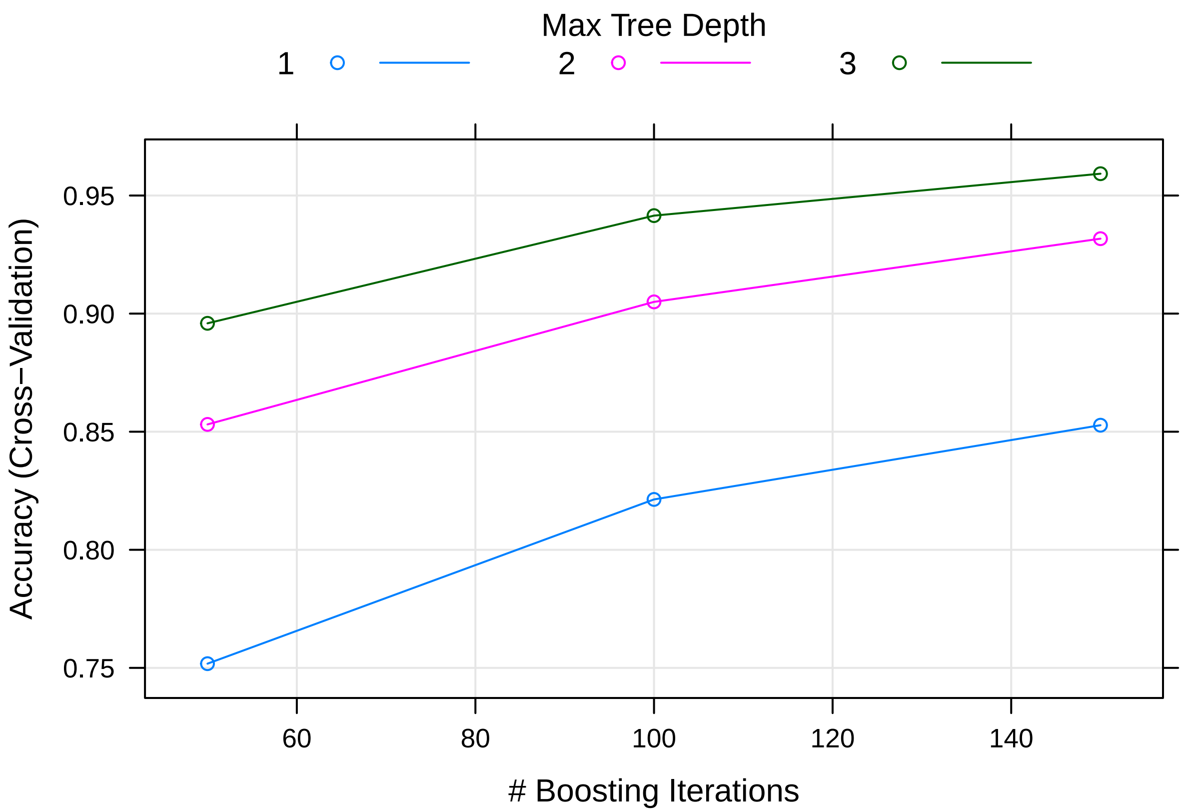
Building the Boosting Model

```
set.seed(12345)
mbst <- train(classe ~., method="gbm", data=training, verbose=F, trControl=trainControl(method="cv", nu
mbst
```

```
## Stochastic Gradient Boosting
##
## 11776 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
```

```
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 10598, 10598, 10598, 10599, 10597, 10599, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7517821  0.6854034
##   1                  100      0.8213293  0.7738550
##   1                  150      0.8527503  0.8136344
##   2                   50      0.8530893  0.8137493
##   2                  100      0.9049759  0.8797193
##   2                  150      0.9317259  0.9136043
##   3                   50      0.8958888  0.8681586
##   3                  100      0.9414919  0.9259695
##   3                  150      0.9592390  0.9484348
##
## 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.
```

```r
plot(mbst)
```



Out-of-sample using confusion matrix

```
pred_bt <- predict(mbst, newdata=testing)
confusionMatrix(pred_bt, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2200   40    0    3    1
##          B   22 1436   39    3   15
##          C    6   37 1314   47   11
##          D    4    3   14 1220   22
##          E    0    2    1   13 1393
##
## Overall Statistics
##
##                Accuracy : 0.9639
##                  95% CI : (0.9596, 0.9679)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9544
##  Mcnemar's Test P-Value : 1.851e-07
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9857   0.9460   0.9605   0.9487   0.9660
## Specificity            0.9922   0.9875   0.9844   0.9934   0.9975
## Pos Pred Value         0.9804   0.9479   0.9286   0.9660   0.9886
## Neg Pred Value         0.9943   0.9870   0.9916   0.9900   0.9924
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2804   0.1830   0.1675   0.1555   0.1775
## Detection Prevalence   0.2860   0.1931   0.1803   0.1610   0.1796
## Balanced Accuracy      0.9889   0.9667   0.9725   0.9711   0.9818
```

Classification of Uknonw test data

Decision Tree Model

```
pred_dT <- predict(mDT2, newdata=test_clean)
pred_dT
```

```
##  [1] A A A A A A A A A A A A A A A A A A A A
## Levels: A B C D E
```

Random forest model

```
pred_rf <- predict(mRF, newdata=test_clean)
pred_rf
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```

Stochastic Gradient Boosting (gbm)

```r
pred_bt <- predict(mbst, newdata=test_clean)
pred_bt
```

```
##  [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
```

I beleive that Random Forest model (n=100) and Stochastic Gradient Boosting (gbm) are very accurate in terms of the out-of-sample accuracy, i.e. 99.44% for Random Forest 96.39% for gbm and ~50% for recursive partiaion tree (rpart) respectively. We can reply on either Random Forest or Stochastic Gradient Boosting (gbm) for the prediction. One caution is that the time it took to build Random Forest is much faster than that for gbm. Therefore, we would decide to use Random Forest to get the perliminary prediction result as much as possible.