L8. Practical machine learning(week4)

Name : Taesoon Kim

Date : Jul-14-2017

Executive summary

In this lecture, I studied the ways how do I analyze data set and find the relationship between variables and result. Dealing with real data, I adjust a variety of prediction method, and forecast the outcome. As a result, when I use random forest, the accuracy is the highest. Thus I apply to test set and get a right result.

Load & check data

```
# set the route
setwd("D:/1-1. R studio/Lecture8. Practical machine learning")

# Load 2 data, training set & test set
train_set<-read.csv(file="pml-training.csv",header=TRUE,sep=",",na.strings=c("NA","#DIV/0!",""))
test_set<-read.csv(file="pml-testing.csv",header=TRUE,sep=",",na.strings=c("NA","#DIV/0!",""))

# Check the column names & data set
library(Hmisc)
```

```
## Loading required package: lattice

## Loading required package: survival

## Loading required package: Formula

## Loading required package: ggplot2

##
## Attaching package: 'Hmisc'

## The following objects are masked from 'package:base':
##
##     format.pval, round.POSIXt, trunc.POSIXt, units
```

```
# describe(train_set)

library(caret)
```

```
##
## Attaching package: 'caret'

## The following object is masked from 'package:survival':
##
##     cluster
```

I read 2 data set, training data and test data, and check the training data set.

Specially, I use describe() function, I can check the number of missing values and unique values in each column. However, when I print the result of describe() function, this report will be very long. So in this report, I mark as comment using "#" symbol.

There are 160 columns and 19,622 rows. However, when I check the training data set, I find a few useless columns, and I think it's better to ignore those columns.

Prepossesing

```
# Delete the column which has only "NA" value
na_col_train_set<-colSums(is.na(train_set))
col_name<-na_col_train_set==0     # store the column which includes "NA" value
col_name_t<-col_name[col_name==TRUE]
rem_name<-names(col_name_t)
new_train<-subset(train_set,select=rem_name)

# Also, delete the "X" and other columns. I don't need
new_train<-subset(new_train,select=-c(X,user_name,raw_timestamp_part_1,raw_timestamp_part_2,cvtd_timest
dim(new_train)
```

```
## [1] 19622    54
```

I exclude "NA" columns, so 100 columns are deleted, and 60 columns are remained. Also, user name & time stamp are not needed, I delete 6 more columns.

Cross validation

```
# Separate the training data and testing data
set.seed(135)
library(caret)     # In order to use createDataPartition() function
inTrain<-createDataPartition(new_train$classe,p=0.6,list=FALSE)
training<-new_train[inTrain,]
testing<-new_train[-inTrain,]
dim(training)
```

```
## [1] 11776    54
```

```
dim(testing)
```

```
## [1] 7846   54
```

I use createDataPartition() function, and separate training data(60%) and testing data(40%). I am going to adjust various modeling methods using training data, since then check the results correct or not using testing data.

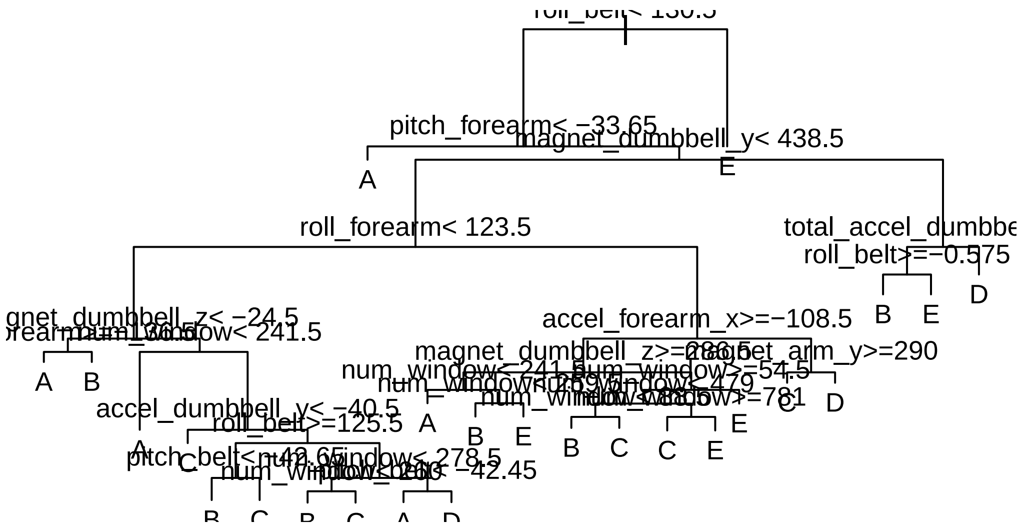
Tree models

1. rpart() function

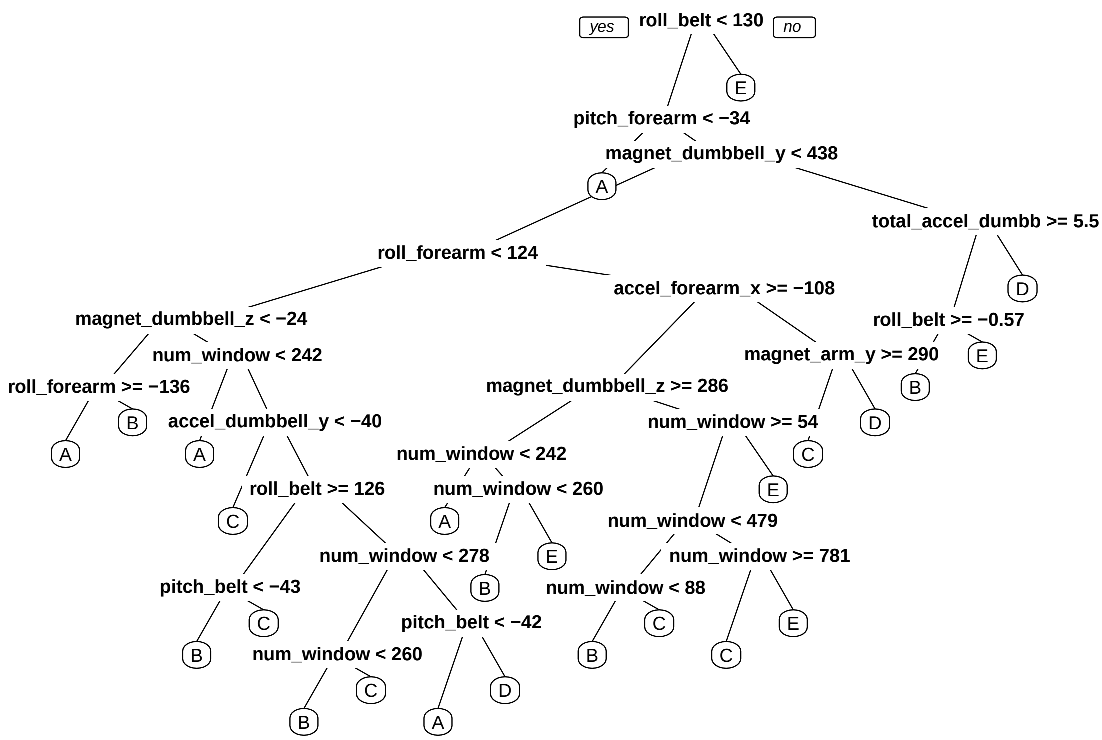
```
# rpart() function
library(rpart)
rpart_training<-rpart(classe~.,data=training,method="class")
rpart_training
```

```
## n= 11776
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##    1) root 11776 8428 A (0.28 0.19 0.17 0.16 0.18)
##      2) roll_belt< 130.5 10764 7421 A (0.31 0.21 0.19 0.18 0.11)
##        4) pitch_forearm< -33.65 979    5 A (0.99 0.0051 0 0 0) *
##        5) pitch_forearm>=-33.65 9785 7416 A (0.24 0.23 0.21 0.2 0.12)
##         10) magnet_dumbbell_y< 438.5 8253 5931 A (0.28 0.18 0.24 0.19 0.1)
##           20) roll_forearm< 123.5 5187 3100 A (0.4 0.18 0.19 0.17 0.06)
##             40) magnet_dumbbell_z< -24.5 1799  613 A (0.66 0.21 0.02 0.082 0.032)
##               80) roll_forearm>=-136.5 1482  329 A (0.78 0.17 0.022 0.026 0.004) *
##               81) roll_forearm< -136.5 317  198 B (0.1 0.38 0.0095 0.35 0.16) *
##             41) magnet_dumbbell_z>=-24.5 3388 2457 C (0.27 0.17 0.27 0.21 0.075)
##               82) num_window< 241.5 796  130 A (0.84 0.0013 0 0.068 0.094) *
##               83) num_window>=241.5 2592 1661 C (0.091 0.22 0.36 0.26 0.069)
##                166) accel_dumbbell_y< -40.5 384   32 C (0 0.044 0.92 0.039 0) *
##                167) accel_dumbbell_y>=-40.5 2208 1550 D (0.11 0.25 0.26 0.3 0.082)
##                  334) roll_belt>=125.5 545  224 C (0 0.36 0.59 0.042 0.0073)
##                    668) pitch_belt< -42.65 213   23 B (0 0.89 0.0047 0.085 0.019) *
##                    669) pitch_belt>=-42.65 332   12 C (0 0.021 0.96 0.015 0) *
##                  335) roll_belt< 125.5 1663 1028 D (0.14 0.22 0.16 0.38 0.11)
##                    670) num_window< 278.5 198   97 C (0 0.49 0.51 0 0)
##                     1340) num_window< 260 97    0 B (0 1 0 0 0) *
##                     1341) num_window>=260 101    0 C (0 0 1 0 0) *
##                    671) num_window>=278.5 1465  830 D (0.16 0.18 0.11 0.43 0.12)
##                     1342) pitch_belt< -42.45 328  209 A (0.36 0.35 0.2 0.079 0.015) *
##                     1343) pitch_belt>=-42.45 1137  528 D (0.1 0.13 0.082 0.54 0.15) *
##           21) roll_forearm>=123.5 3066 2047 C (0.077 0.17 0.33 0.24 0.18)
##             42) accel_forearm_x>=-108.5 2179 1355 C (0.083 0.21 0.38 0.11 0.21)
##               84) magnet_dumbbell_z>=286.5 548  388 A (0.29 0.28 0.018 0.16 0.25)
##                168) num_window< 241.5 118    0 A (1 0 0 0 0) *
##                169) num_window>=241.5 430  276 B (0.098 0.36 0.023 0.2 0.32)
##                  338) num_window< 259.5 131    0 B (0 1 0 0 0) *
##                  339) num_window>=259.5 299  162 E (0.14 0.077 0.033 0.29 0.46) *
##               85) magnet_dumbbell_z< 286.5 1631  817 C (0.013 0.19 0.5 0.097 0.2)
##                170) num_window>=54.5 1514  700 C (0.014 0.2 0.54 0.11 0.14)
##                  340) num_window< 479 1035  355 C (0.017 0.22 0.66 0.1 0.0087)
##                    680) num_window< 88.5 99    5 B (0.051 0.95 0 0 0) *
##                    681) num_window>=88.5 936  256 C (0.014 0.14 0.73 0.11 0.0096) *
##                  341) num_window>=479 479  277 E (0.0063 0.18 0.28 0.12 0.42)
##                    682) num_window>=781 262  128 C (0.011 0.32 0.51 0.088 0.069) *
##                    683) num_window< 781 217   33 E (0 0 0 0.15 0.85) *
##                171) num_window< 54.5 117    0 E (0 0 0 0 1) *
##             43) accel_forearm_x< -108.5 887  403 D (0.061 0.082 0.22 0.55 0.091)
##               86) magnet_arm_y>=290 273  121 C (0.048 0.1 0.56 0.23 0.062) *
##               87) magnet_arm_y< 290 614  193 D (0.067 0.073 0.07 0.69 0.1) *
##         11) magnet_dumbbell_y>=438.5 1532  739 B (0.031 0.52 0.044 0.21 0.2)
##           22) total_accel_dumbbell>=5.5 1117  389 B (0.042 0.65 0.06 0.021 0.22)
##             44) roll_belt>=-0.575 947  219 B (0.05 0.77 0.071 0.025 0.086) *
##             45) roll_belt< -0.575 170    0 E (0 0 0 0 1) *
##           23) total_accel_dumbbell< 5.5 415  114 D (0 0.16 0.0024 0.73 0.12) *
##      3) roll_belt>=130.5 1012    5 E (0.0049 0 0 0 1) *
```

```r
plot(rpart_training,compress=TRUE)
text(rpart_training,cex=0.8)
```




```r
# Prettier graph
library(rpart.plot)
prp(rpart_training)
```




```r
# Predict this tree is right or not
# I am going to use predict() function and confusionMatrix() function
library(e1071)
```

```
##
## Attaching package: 'e1071'

## The following object is masked from 'package:Hmisc':
##
##     impute
```

```r
rpart_prediction<-predict(rpart_training,testing,type="class")
confusionMatrix(rpart_prediction,testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1993  261   53   82   67
##          B   63  860   42   81   75
##          C   16  217 1178  122   35
##          D  123  161   89  931  208
##          E   37   19    6   70 1057
##
## Overall Statistics
##
##                Accuracy : 0.7671
##                  95% CI : (0.7576, 0.7765)
```

```
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7049
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8929   0.5665   0.8611   0.7240   0.7330
## Specificity            0.9175   0.9588   0.9398   0.9114   0.9794
## Pos Pred Value         0.8115   0.7672   0.7513   0.6157   0.8890
## Neg Pred Value         0.9557   0.9022   0.9697   0.9440   0.9422
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2540   0.1096   0.1501   0.1187   0.1347
## Detection Prevalence   0.3130   0.1429   0.1998   0.1927   0.1515
## Balanced Accuracy      0.9052   0.7626   0.9005   0.8177   0.8562
```

I predict using rpart() function. When I draw a tree using plot() function, a little bit hard to analyze. Therefore I use prp() function, tree graph is much prettier and easy to look at.

Using rpart() function, the accuracy is **76.7%**. So I need to adjust another predictive method.

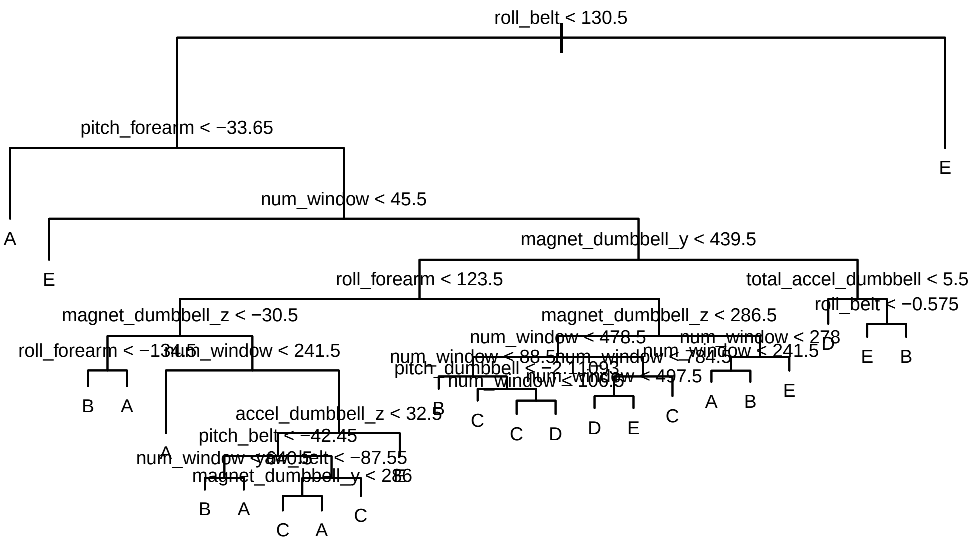
2. tree() function

```r
# tree() function & draw a graph
library(tree)
tree_training<-tree(classe~.,data=training)
tree_training
```

```
## node), split, n, deviance, yval, (yprob)
##      * denotes terminal node
##
##    1) root 11776 37400.00 A ( 0.284307 0.193529 0.174423 0.163893 0.183849 )
##      2) roll_belt < 130.5 10764 33500.00 A ( 0.310572 0.211724 0.190821 0.179301 0.107581 )
##        4) pitch_forearm < -33.65 979    62.75 A ( 0.994893 0.005107 0.000000 0.000000 0.000000 ) *
##        5) pitch_forearm > -33.65 9785 30980.00 A ( 0.242105 0.232397 0.209913 0.197241 0.118344 )
##         10) num_window < 45.5 436   444.00 E ( 0.000000 0.000000 0.000000 0.206422 0.793578 ) *
##         11) num_window > 45.5 9349 29110.00 A ( 0.253396 0.243235 0.219703 0.196812 0.086854 )
##           22) magnet_dumbbell_y < 439.5 7949 24450.00 A ( 0.292364 0.186816 0.249969 0.190590 0.0802
##             44) roll_forearm < 123.5 5051 14450.00 A ( 0.413383 0.187884 0.191645 0.162344 0.044744 )
##               88) magnet_dumbbell_z < -30.5 1654  3000.00 A ( 0.678960 0.215840 0.010278 0.079202 0.0
##                176) roll_forearm < -134.5 276   678.40 B ( 0.123188 0.423913 0.010870 0.369565 0.0724
##                177) roll_forearm > -134.5 1378  1769.00 A ( 0.790276 0.174165 0.010160 0.021045 0.004
##               89) magnet_dumbbell_z > -30.5 3397 10250.00 A ( 0.284074 0.174271 0.279953 0.202826 0.0
##                178) num_window < 241.5 715   214.50 A ( 0.970629 0.001399 0.000000 0.004196 0.023776
##                179) num_window > 241.5 2682  7855.00 C ( 0.101044 0.220358 0.354586 0.255779 0.06823
##                  358) accel_dumbbell_z < 32.5 2292  6189.00 C ( 0.117801 0.194590 0.414485 0.257417 0
##                    716) pitch_belt < -42.45 548  1299.00 B ( 0.220803 0.560219 0.122263 0.080292 0.0
##                     1432) num_window < 640.5 373   681.60 B ( 0.000000 0.678284 0.179625 0.117962 0.0
##                     1433) num_window > 640.5 175   216.30 A ( 0.691429 0.308571 0.000000 0.000000 0.0
##                    717) pitch_belt > -42.45 1744  4131.00 C ( 0.085436 0.079702 0.506307 0.313073 0.0
##                     1434) yaw_belt < -87.55 557  1610.00 C ( 0.254937 0.231598 0.312388 0.177738 0.0
##                       2868) magnet_dumbbell_y < 286 315   664.90 C ( 0.022222 0.336508 0.552381 0.04
```

```
##                       2869) magnet_dumbbell_y > 286 242   443.60 A ( 0.557851 0.095041 0.000000 0.34
##                     1435) yaw_belt > -87.55 1187  1896.00 C ( 0.005897 0.008425 0.597304 0.376580 0.0
##                  359) accel_dumbbell_z > 32.5 390   866.80 E ( 0.002564 0.371795 0.002564 0.246154 0
##             45) roll_forearm > 123.5 2898  8715.00 C ( 0.081435 0.184955 0.351622 0.239821 0.142167 )
##               90) magnet_dumbbell_z < 286.5 2309  6296.00 C ( 0.032915 0.150715 0.435253 0.262884 0.1
##                180) num_window < 478.5 1475  3394.00 C ( 0.045424 0.173559 0.572881 0.190508 0.01762
##                  360) num_window < 88.5 125   103.00 B ( 0.144000 0.856000 0.000000 0.000000 0.00000
##                  361) num_window > 88.5 1350  2861.00 C ( 0.036296 0.110370 0.625926 0.208148 0.0192
##                    722) pitch_dumbbell < -2.11093 826  1325.00 C ( 0.059322 0.129540 0.756659 0.0544
##                    723) pitch_dumbbell > -2.11093 524  1127.00 D ( 0.000000 0.080153 0.419847 0.4503
##                     1446) num_window < 106.5 181     0.00 C ( 0.000000 0.000000 1.000000 0.000000 0.0
##                     1447) num_window > 106.5 343   656.60 D ( 0.000000 0.122449 0.113703 0.688047 0.0
##                181) num_window > 478.5 834  2229.00 D ( 0.010791 0.110312 0.191847 0.390887 0.296163
##                  362) num_window < 784.5 497   684.50 D ( 0.000000 0.000000 0.000000 0.547284 0.4527
##                    724) num_window < 497.5 220     0.00 D ( 0.000000 0.000000 0.000000 1.000000 0.00
##                    725) num_window > 497.5 277   267.50 E ( 0.000000 0.000000 0.000000 0.187726 0.81
##                  363) num_window > 784.5 337   860.30 C ( 0.026706 0.272997 0.474777 0.160237 0.0652
##               91) magnet_dumbbell_z > 286.5 589  1687.00 B ( 0.271647 0.319185 0.023769 0.149406 0.2
##                182) num_window < 278 291   490.20 B ( 0.405498 0.546392 0.048110 0.000000 0.000000 )
##                  364) num_window < 241.5 118     0.00 A ( 1.000000 0.000000 0.000000 0.000000 0.0000
##                  365) num_window > 241.5 173    97.23 B ( 0.000000 0.919075 0.080925 0.000000 0.0000
##                183) num_window > 278 298   726.40 E ( 0.140940 0.097315 0.000000 0.295302 0.466443 )
##           23) magnet_dumbbell_y > 439.5 1400  3297.00 B ( 0.032143 0.563571 0.047857 0.232143 0.1242
##             46) total_accel_dumbbell < 5.5 415   653.50 D ( 0.000000 0.156627 0.002410 0.725301 0.11
##             47) total_accel_dumbbell > 5.5 985  1777.00 B ( 0.045685 0.735025 0.067005 0.024365 0.12
##               94) roll_belt < -0.575 90     0.00 E ( 0.000000 0.000000 0.000000 0.000000 1.000000 )
##               95) roll_belt > -0.575 895  1325.00 B ( 0.050279 0.808939 0.073743 0.026816 0.040223 )
##     3) roll_belt > 130.5 1012    63.08 E ( 0.004941 0.000000 0.000000 0.000000 0.995059 ) *
```

```r
plot(tree_training)
text(tree_training,cex=0.6)
```




```r
# Predict testing values
tree_prediction<-predict(tree_training,testing,type="class")
confusionMatrix(tree_prediction,testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2075  232    9   91   22
##          B   74  868   82   86   24
##          C   45  222 1223  390   35
##          D    1   86   52  500   45
##          E   37  110    2  219 1316
##
## Overall Statistics
##
##                Accuracy : 0.7624
##                  95% CI : (0.7528, 0.7718)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6984
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
```

```
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9297   0.5718   0.8940  0.38880   0.9126
## Specificity            0.9369   0.9580   0.8932  0.97195   0.9425
## Pos Pred Value         0.8543   0.7654   0.6386  0.73099   0.7815
## Neg Pred Value         0.9710   0.9032   0.9756  0.89025   0.9796
## Prevalence             0.2845   0.1935   0.1744  0.16391   0.1838
## Detection Rate         0.2645   0.1106   0.1559  0.06373   0.1677
## Detection Prevalence   0.3096   0.1445   0.2441  0.08718   0.2146
## Balanced Accuracy      0.9333   0.7649   0.8936  0.68038   0.9276
```

I predict using tree() function. Accuracy is **76.2%**, and it's less than using rpart() function. Thus I will consider another way.

3. randomForest() function

```
# randomForest() function
library(randomForest)
```

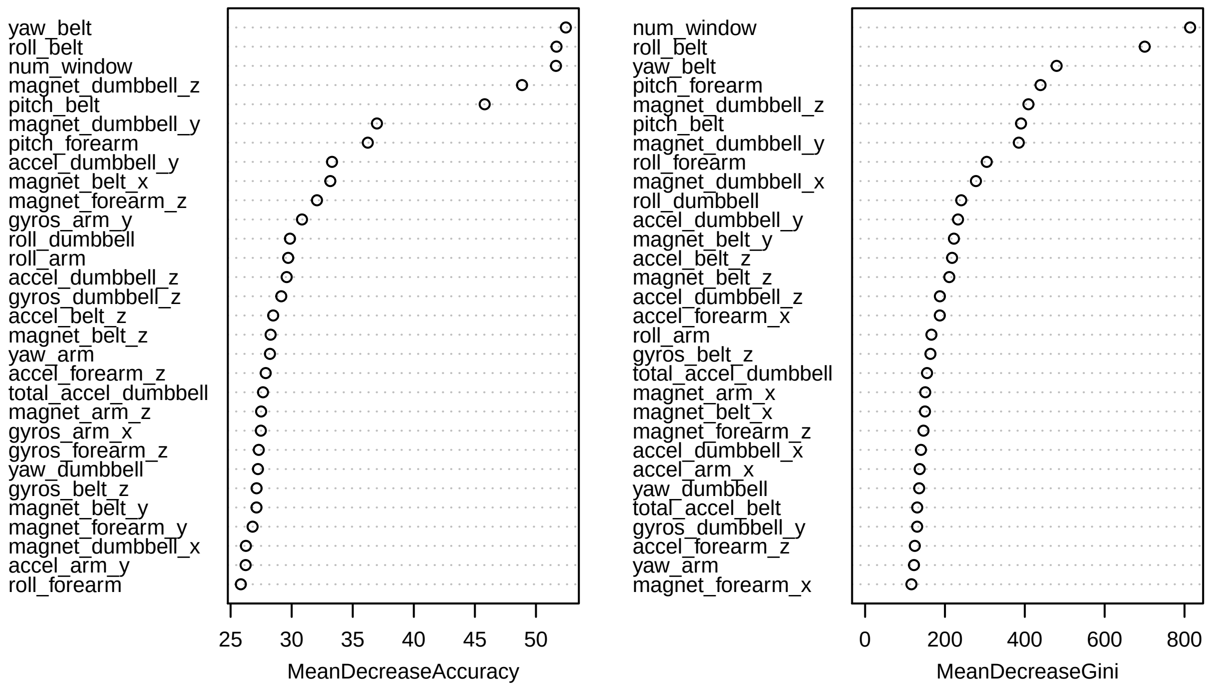
```
## randomForest 4.6-12

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:Hmisc':
##
##     combine

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

```
rf_training<-randomForest(classe~.,data=training,method="class",importance=TRUE)
rf_training
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = training, method = "class",      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.34%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3347    1    0    0    0 0.0002986858
## B    3 2272    4    0    0 0.0030715226
## C    0    8 2044    2    0 0.0048685492
## D    0    0   14 1915    1 0.0077720207
## E    0    0    0    7 2158 0.0032332564
```

```
# draw a graph
varImpPlot(rf_training,main="varImpPlot of train data",cex=0.7)
```

varImpPlot of train data




```
# check the accuracy
rf_prediction<-predict(rf_training,testing,type="class")
confusionMatrix(rf_prediction,testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    3    0    0    0
##          B    0 1513    3    0    0
##          C    0    2 1365    4    0
##          D    0    0    0 1281    0
##          E    0    0    0    1 1442
##
## Overall Statistics
##
##                Accuracy : 0.9983
##                  95% CI : (0.9972, 0.9991)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9979
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
```

```
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9967   0.9978   0.9961   1.0000
## Specificity            0.9995   0.9995   0.9991   1.0000   0.9998
## Pos Pred Value         0.9987   0.9980   0.9956   1.0000   0.9993
## Neg Pred Value         1.0000   0.9992   0.9995   0.9992   1.0000
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1928   0.1740   0.1633   0.1838
## Detection Prevalence   0.2849   0.1932   0.1747   0.1633   0.1839
## Balanced Accuracy      0.9997   0.9981   0.9984   0.9981   0.9999
```

When I use random forest method, the accuracy is **99.8%**, almost 100%. So I think random forest is the best way, and apply to real test set.

Conclusion

Let's look at the accuracy - rpart : 76.7% - tree : 76.2% - random forest : 99.8%

```r
answer<-predict(rf_training,test_set,type="class")
answer
```

```
##  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
```

I get a result.