Practical Machine Learning Project

Zhimao He

Introduction

The goal of this project is to predict the manner the participants did the exercise, which is the “classe” variable in the training set. This is the report for how we conclude the model, how we cross validate the model, what the expected out of sample error is for the model. The final model will be used to predict 20 test cases.

About the data

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

Data analysis

First, we load the data, and explore the data and see what is useful to build our data and what is not so interesting. #### Read teh data

```
## Reading the data
training = read.csv(url("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-trainin
g.csv"))
validation = read.csv(url("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testi
ng.csv"))
```

Exploring the data

Since we find out that there were some empty and junk data, we reload the data and replace they with NA.

```
## Look at the data structure
str(training, list.len=ncol(training))
## Reading the data
training = read.csv(url("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-trainin
g.csv"), na.strings=c("NA","#DIV/0!",""))
validation = read.csv(url("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testi
ng.csv"), na.strings=c("NA","#DIV/0!",""))
summary(training)
```

Cleaning the data

From the summary, we can see that, there are a lot of variables, in fact 160 of them in the data. A lot of variables could introduce noise and some of them might not have coorlation Thre are some columns that we don’t care for the model, such as the row count, user name, time stamp. Also, some of the columns has a lot of NA which could be excluded for our model.

```
## Get rid of some columns that we don't care.
cleanTraining <- training[ , -which(names(training) %in% c("X","user_name", "raw_timesta
mp_part_1","raw_timestamp_part_2","cvtd_timestamp", "new_window","num_window"))]

## Get rid of columns that contains mostly NA, for 95% and up is NA
cleanTraining <- cleanTraining[,apply(cleanTraining, 2, function(col) (sum(is.na(col))/l
ength(col))< 0.95)]
str(cleanTraining)

cleanColNames <- colnames(cleanTraining)
cleanValidation <- validation[cleanColNames[-53]]
str(cleanValidation)
```

We are using the format that if a column has 95% and more of the NA values, we exclude the column. Now we are down to 53 variables. Now let's split the data to training and testing using 60% for training, 40% for testing and fit some models. #### Get training and testing sets

```
## Load library used
suppressMessages(library(caret))
```

```
## Warning: package 'caret' was built under R version 3.2.5
```

```
## Warning: package 'ggplot2' was built under R version 3.2.4
```

```
suppressMessages(library(rpart))
#library(rattle)
suppressMessages(library(randomForest))
suppressMessages(library(dplyr))
suppressMessages(library(reshape2))

## Set seeds
set.seed(89999)

## Split training/test
inTrain <- createDataPartition(cleanTraining$classe, p=0.6, list=FALSE)
actTraining <- cleanTraining[inTrain, ]
actTesting <- cleanTraining[-inTrain, ]
```

Before fitting a model, we would like to know what variables might coolaete with classe. #### Find out what variable is correlate with each other

```
temp <- actTraining
temp$classe <- as.numeric(temp$classe)
corlMatrix<- data.frame(cor(temp))
corlMatrix$name <- names(temp[1:53])
corlEnd <- data.frame(cbind(corlMatrix$classe, corlMatrix$name))
names(corlEnd) <- c("cor", "name")
tail(arrange(corlEnd,cor), 13)
```

```
##                    cor                name
## 41 0.0670569592096512           roll_belt
## 42 0.0695055646460204   magnet_dumbbell_x
## 43 0.0762988715510022    accel_dumbbell_z
## 44 0.0821433499368408    total_accel_belt
## 45 0.0846972568464369      pitch_dumbbell
## 46 0.0852867436767401            roll_arm
## 47  0.123029689470699    accel_dumbbell_x
## 48  0.143512517833186   magnet_dumbbell_z
## 49  0.151980251638139 total_accel_forearm
## 50  0.234767491109343         accel_arm_x
## 51    0.2869163751345        magnet_arm_x
## 52   0.34304766973813       pitch_forearm
## 53                  1              classe
```

We could use some of these variables to fit the model, top 12 are roll_belt, magnet_dumbbell_x, accel_dumbbell_z, roll_arm, total_accel_belt, pitch_dumbbell, accel_dumbbell_x, total_accel_forearm, magnet_dumbbell_z, accel_arm_x, magnet_arm_x and pitch_forearm.

Fitting models

Let us try to fit differnt models. Since the data has a lot of variables and also they seems to have catergories, regression tree seems to be a good choice. We could try that.

Regression tree

```
treeMod <- train(classe~., data = actTraining, method="rpart")
print(treeMod$finalModel)
```

```
## 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< 129.5 10698 7389 A (0.31 0.21 0.19 0.18 0.11)
##      4) pitch_forearm< -33.15 962   13 A (0.99 0.014 0 0 0) *
##      5) pitch_forearm>=-33.15 9736 7376 A (0.24 0.23 0.21 0.2 0.12)
##       10) magnet_dumbbell_y< 439.5 8222 5903 A (0.28 0.18 0.24 0.19 0.1)
##         20) roll_forearm< 120.5 5050 2987 A (0.41 0.18 0.19 0.17 0.055) *
##         21) roll_forearm>=120.5 3172 2148 C (0.081 0.18 0.32 0.24 0.18) *
##       11) magnet_dumbbell_y>=439.5 1514  737 B (0.027 0.51 0.044 0.22 0.19) *
##    3) roll_belt>=129.5 1078   39 E (0.036 0 0 0 0.96) *
```

```
# fancyRpartPlot(treeMod$finalModel)
treePred <- predict(treeMod, newdata=actTesting)
treeResult<-confusionMatrix(actTesting$classe, treePred)
## Importance
varImp(treeMod)
```

```
## rpart variable importance
##
##   only 20 most important variables shown (out of 52)
##
##                    Overall
## pitch_forearm      100.00
## roll_belt           90.82
## roll_forearm        71.03
## magnet_dumbbell_y   50.76
## accel_belt_z        43.57
## yaw_belt            40.62
## magnet_belt_y       40.47
## total_accel_belt    35.89
## roll_dumbbell       34.52
## magnet_arm_x        26.50
## accel_arm_x         24.54
## magnet_dumbbell_z   19.38
## accel_dumbbell_y    16.35
## gyros_belt_y         0.00
## pitch_dumbbell       0.00
## magnet_belt_z        0.00
## accel_arm_y          0.00
## pitch_belt           0.00
## gyros_dumbbell_x     0.00
## magnet_arm_z         0.00
```

```
treeResult
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1995   40  162    0   35
##          B  621  509  388    0    0
##          C  620   41  707    0    0
##          D  576  228  482    0    0
##          E  212  177  393    0  660
##
## Overall Statistics
##
##                Accuracy : 0.4934
##                  95% CI : (0.4823, 0.5045)
##     No Information Rate : 0.5129
##     P-Value [Acc > NIR] : 0.9997
##
##                   Kappa : 0.3385
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.4958  0.51156  0.33161       NA  0.94964
## Specificity            0.9380  0.85272  0.88432   0.8361  0.89064
## Pos Pred Value         0.8938  0.33531  0.51681       NA  0.45770
## Neg Pred Value         0.6386  0.92320  0.78002       NA  0.99453
## Prevalence             0.5129  0.12682  0.27173   0.0000  0.08858
## Detection Rate         0.2543  0.06487  0.09011   0.0000  0.08412
## Detection Prevalence   0.2845  0.19347  0.17436   0.1639  0.18379
## Balanced Accuracy      0.7169  0.68214  0.60797       NA  0.92014
```

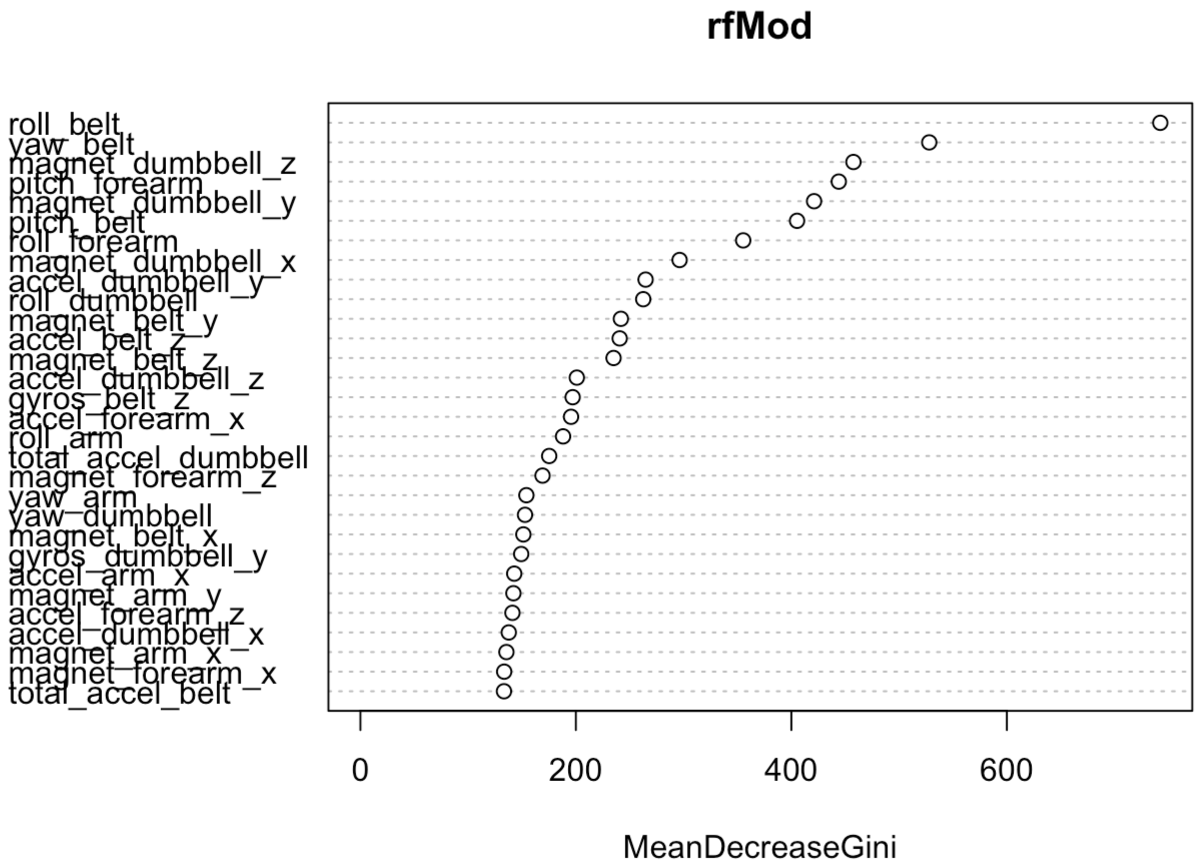
The accuracy is of regression tree is only 0.4869. So this is not a very good model. Let's try random forest, since it is usually quite accurate but could be over fitting.

Ramdom forest

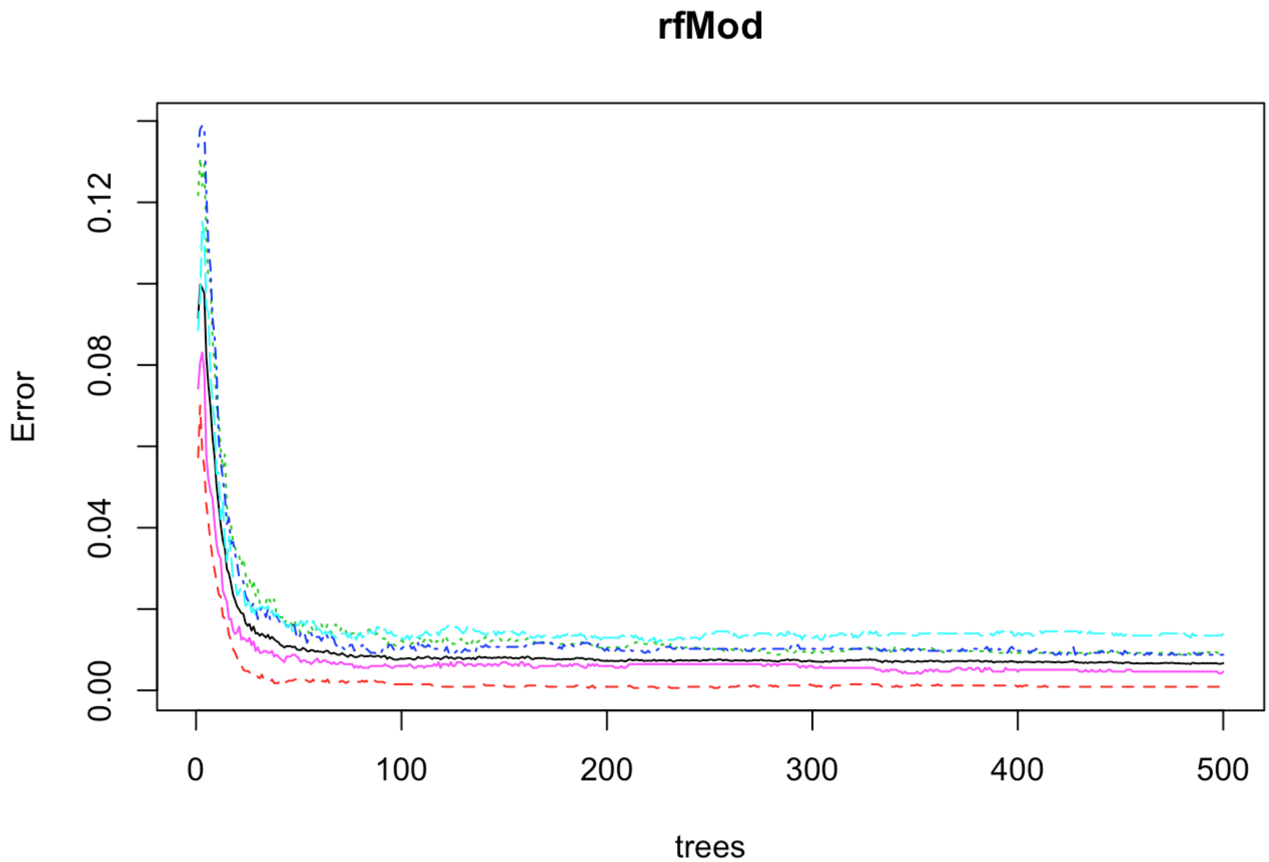
```
## Model
rfMod <- randomForest(classe ~., data=actTraining)
## Predict
rfPred <- predict(rfMod,actTesting)
## Accuracy
rfResult<- confusionMatrix(actTesting$classe, rfPred)
rfResult
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 2230    2    0    0    0
##          B   10 1496   12    0    0
##          C    0    6 1361    1    0
##          D    0    0   21 1265    0
##          E    0    0    0    3 1439
## 
## Overall Statistics
##                                           
##                Accuracy : 0.993           
##                  95% CI : (0.9909, 0.9947)
##     No Information Rate : 0.2855          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.9911          
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9955   0.9947   0.9763   0.9968   1.0000
## Specificity            0.9996   0.9965   0.9989   0.9968   0.9995
## Pos Pred Value         0.9991   0.9855   0.9949   0.9837   0.9979
## Neg Pred Value         0.9982   0.9987   0.9949   0.9994   1.0000
## Prevalence             0.2855   0.1917   0.1777   0.1617   0.1834
## Detection Rate         0.2842   0.1907   0.1735   0.1612   0.1834
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9976   0.9956   0.9876   0.9968   0.9998
```

```
## Importance
varImpPlot(rfMod)
```


rfMod
roll_belt
yaw_belt
magnet_dumbbell_z
pitch_forearm
magnet_dumbbell_y
pitch_belt
roll_forearm
magnet_dumbbell_x
accel_dumbbell_y
roll_dumbbell
magnet_belt_y
accel_belt_z
magnet_belt_z
accel_dumbbell_z
gyros_belt_z
accel_forearm_x
roll_arm
total_accel_dumbbell
magnet_forearm_z
yaw_arm
yaw_dumbbell
magnet_belt_x
gyros_dumbbell_y
accel_arm_x
magnet_arm_y
accel_forearm_z
accel_dumbbell_x
magnet_arm_x
magnet_forearm_x
total_accel_belt
0
200
400
600
MeanDecreaseGini

```
plot(rfMod)
```



The accuracy looks great, it is 0.993. And looking at the prediction table.

Let’s see if by using few variables, if there is any indication of over fitting.

```
rfSmallMod <- randomForest(classe ~ roll_belt + magnet_dumbbell_x + accel_dumbbell_z + r
oll_arm + total_accel_belt + pitch_dumbbell + accel_dumbbell_x + total_accel_forearm + m
agnet_dumbbell_z + accel_arm_x + magnet_arm_x + pitch_forearm, data = actTraining)

rfSmallPred <- predict(rfSmallMod,actTesting)
rfSmallResult<- confusionMatrix(actTesting$classe, rfSmallPred)
rfSmallResult
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2217    8    4    3    0
##          B   23 1442   46    5    2
##          C    0   27 1334    7    0
##          D    1    1   51 1233    0
##          E    0    9   11    8 1414
##
## Overall Statistics
##
##                Accuracy : 0.9737
##                  95% CI : (0.97, 0.9772)
##     No Information Rate : 0.2856
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9668
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9893   0.9697   0.9225   0.9817   0.9986
## Specificity            0.9973   0.9880   0.9947   0.9920   0.9956
## Pos Pred Value         0.9933   0.9499   0.9751   0.9588   0.9806
## Neg Pred Value         0.9957   0.9929   0.9827   0.9965   0.9997
## Prevalence             0.2856   0.1895   0.1843   0.1601   0.1805
## Detection Rate         0.2826   0.1838   0.1700   0.1572   0.1802
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9933   0.9789   0.9586   0.9868   0.9971
```

This fit is also pretty good, it has the accuracy of 0.9709. But seems like using all varibles does improve the accuracy. Well, we could try another fitting, I assume it won't get better accuracy than random forest.

Boosted regression

```
modControl <- trainControl(method = "repeatedcv", number = 6, repeats = 1)
boostedMod<- train(classe ~ ., data=actTraining, method = "gbm", trControl = modControl,
 verbose = FALSE)
```

```
## Loading required package: gbm
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

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

```
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.1
```

```
## Loading required package: plyr
```

```
## -------------------------------------------------------------------------
```

```
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
```

```
## -------------------------------------------------------------------------
```

```
##
## Attaching package: 'plyr'
```

```
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
boostedPre <- predict(boostedMod, newdata=actTesting)
boostedResult <- confusionMatrix(actTesting$classe, boostedPre)
boostedResult
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 2207   18    4    3    0
##          B   52 1414   46    5    1
##          C    0   40 1312   12    4
##          D    2    2   48 1226    8
##          E    4   16    4   30 1388
## 
## Overall Statistics
##                                           
##                Accuracy : 0.9619          
##                  95% CI : (0.9574, 0.966) 
##     No Information Rate : 0.2887          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.9518          
##  Mcnemar's Test P-Value : 7.476e-12       
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9744   0.9490   0.9279   0.9608   0.9907
## Specificity            0.9955   0.9836   0.9913   0.9909   0.9916
## Pos Pred Value         0.9888   0.9315   0.9591   0.9533   0.9626
## Neg Pred Value         0.9897   0.9880   0.9843   0.9924   0.9980
## Prevalence             0.2887   0.1899   0.1802   0.1626   0.1786
## Detection Rate         0.2813   0.1802   0.1672   0.1563   0.1769
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9850   0.9663   0.9596   0.9758   0.9912
```

Booting's accuracy is also pretty good, 0.9638 though it is not as good as random forest.

Cross validation

Let's try to use random sampling to do cross validation. We split the training set itself up, into training and test sets over and over again. Keep rebuilding our models, and picking the one that works best on the test set. Here, we will resample 10 time and see which data set is the best.

```r
set.seed(3000)

## Get total number of data
totalNumber <- nrow(cleanTraining)
## Training is always 60 percent
trainNumber <- round(totalNumber * 0.6)
testNumber <- totalNumber - trainNumber;

crossValidateResult <- as.data.frame(matrix(nrow=7, ncol=10))
modelAccuracy <- vector();

## Do this 10 times
for (i in 1:10){
  rowIndexes = sample(nrow(cleanTraining), trainNumber)
  sampleTraining <- cleanTraining[rowIndexes,]
  sampleTesting <- cleanTraining[-rowIndexes,]

  sampleMod <- randomForest(classe~., data=sampleTraining)
  samplePre <-  predict(sampleMod, sampleTesting)
  sampleResult <- confusionMatrix(sampleTesting$classe, samplePre)

  modelAccuracy[i] <- sampleResult$overall["Accuracy"]
  crossValidateResult[,i] <- sampleResult$overall
  print(sampleResult$overall)
}
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9924831      0.9904891      0.9903143      0.9942730      0.2862785
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9921009      0.9900002      0.9898850      0.9939386      0.2876800
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9946490      0.9932484      0.9927738      0.9961408      0.2753217
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9931201      0.9912890      0.9910327      0.9948275      0.2880622
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9924831      0.9904801      0.9903143      0.9942730      0.2902281
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9931201      0.9912949      0.9910327      0.9948275      0.2865333
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9908269      0.9883701      0.9884617      0.9928159      0.2930310
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9938846      0.9922603      0.9918999      0.9954876      0.2869155
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9929927      0.9911349      0.9908887      0.9947169      0.2853867
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9929927      0.9911297      0.9908887      0.9947169      0.2881896
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

Out of sample error

```
## Get out of sample error
ofsError = 1- mean(modelAccuracy)
ofsError
```

```
## [1] 0.007134667
```

The out of sample error is the average error of 10 samples.

Conclusion

Looking at different errors including sensitivity, specificity, random forest works the best among regression tree and boosting. The accuracy for random forest model is very high for the testing set. Also, the model indicates that most vairlibes are highly correlated with classe. Random forest model usually yields high accurate, however, data process time takes longer. Using less variables by selecting variables taht correlates to classe the most for ramdom forest model could speed up the process that could still have a reasonable good prediciton.