Practical ML Project

Grace

12/9/2016

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(rpart)
library(rpart.plot)
suppressMessages(library(randomForest))
library(caret)
```

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

```
library(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
```

Introduction to the Project

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

Data

The training data for this project are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv) The test data are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)

Goal

The goal of your project is to predict the manner in which they did the exercise. This is the “classe” variable in the training set. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

Loading data

```
setwd("/Users/Grace/Desktop/Data Science/Practical ML")
training <- read.csv("pml-training.csv",na.strings=c("NA","","#DIV/0!"))
testing <- read.csv("pml-testing.csv",na.strings=c("NA","","#DIV/0!"))
```

Split data into training and testing set

```
set.seed(123)
inTrain <- createDataPartition(training$class,p=.67,list=FALSE)
train <- training[inTrain,]
test <- training[-inTrain,]
dim(train);dim(test)
```

```
## [1] 13148   160
```

```
## [1] 6474  160
```

Get a sense of the dataset

```
dim(train)
```

```
## [1] 13148   160
```

```
vars <- names(train) %in% c("classe","new_window","user_name")
sapply(train[vars],summary)
```

```
## $user_name
##   adelmo carlitos  charles   eurico   jeremy    pedro 
##     2621     2065     2350     2106     2296     1710 
## 
## $new_window
##    no   yes 
## 12865   283 
## 
## $classe
##    A    B    C    D    E 
## 3739 2544 2293 2155 2417
```

Data Cleaninig

Remove Near Zero Variance vars nearZeroVar diagnoses predictors that have 1) one unique value 2)few unique values relative to the number of sample 3) ratio of the frequency of the most common value to the second common value is large Since there are 160 predictors in the sample, it is very necessary to remove variables with poor quality.

```
train <- train[,-1] #remove the first column X
nzvars <- nearZeroVar(train,saveMetric=TRUE)
train <- train[,nzvars$nzv==FALSE]
dim(train)
```

```
## [1] 13148   126
```

Remove columns with over 60% NAs

```
M <- sapply(train, function(x) sum(is.na(x))/length(x))
trainc <-train[names(train)[M<0.6]]
var <- names(train)[M<0.6]

testc<- test[var] # select the same variables
testing <- testing[var[-58]] #select the same variable except classe
dim(trainc)
```

```
## [1] 13148    58
```

```
dim(testc)
```

```
## [1] 6474   58
```

```
dim(testing)
```

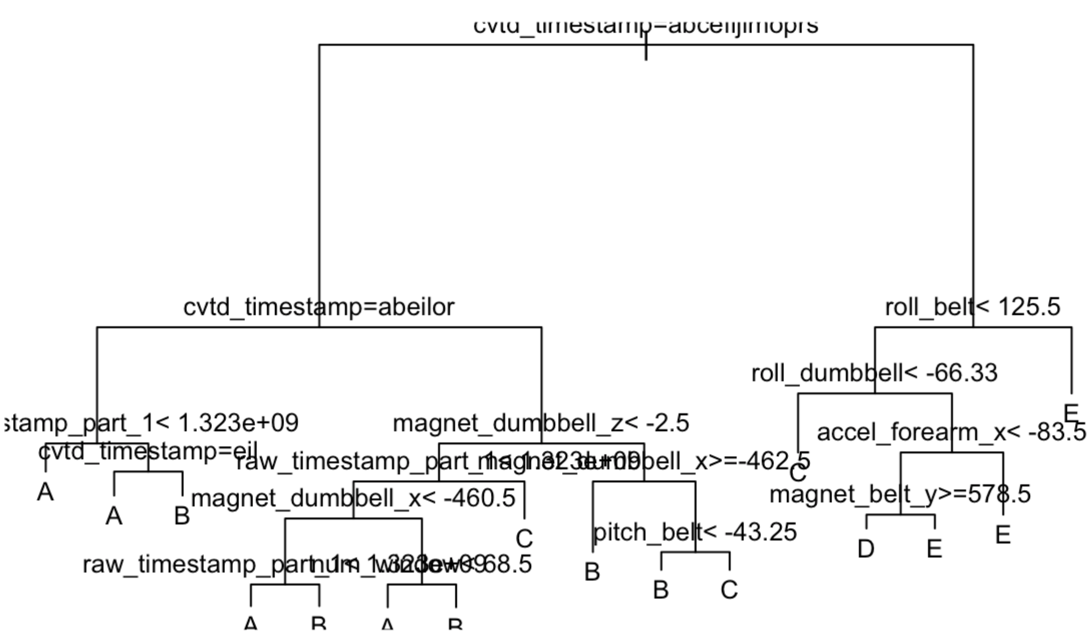
```
## [1] 20 57
```

Model Fitting

Decision Trees

It is a typical classification problem; therefore, I will start with an easy method, decision trees, to extract important features

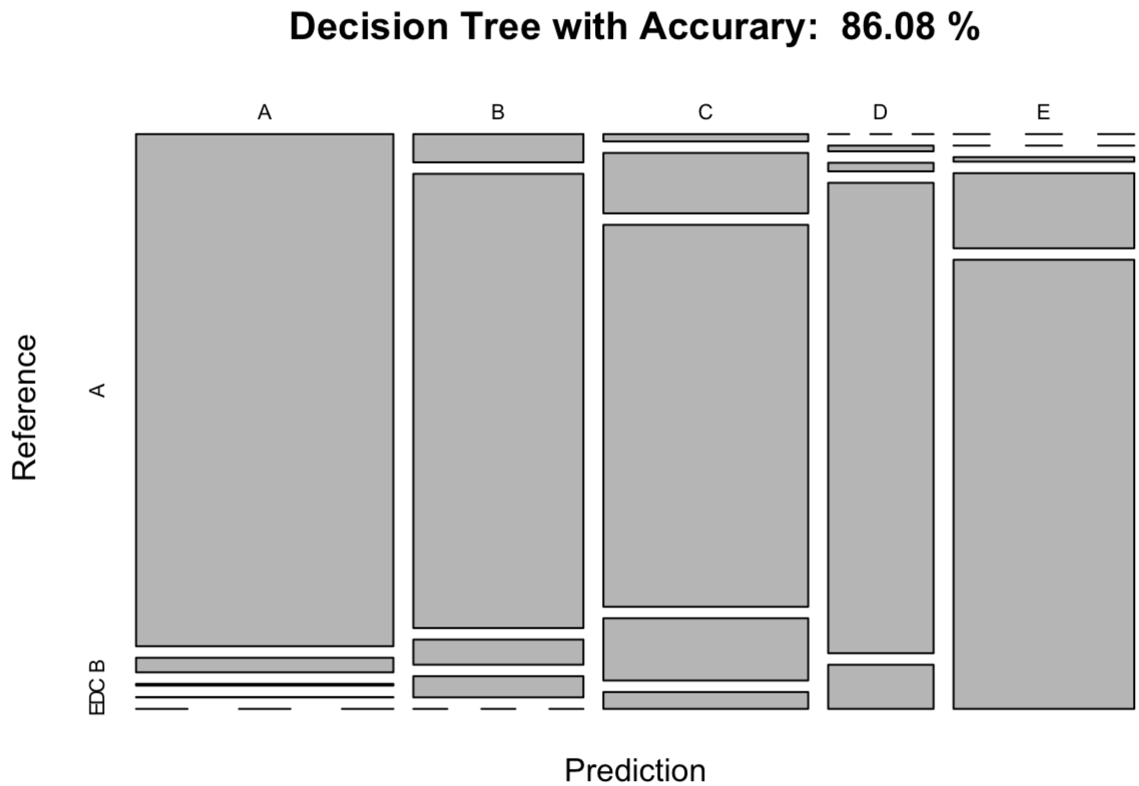
```
set.seed(123)
mod.tree <- rpart(classe~.,data=trainc,method="class")
plot(mod.tree)
text(mod.tree, cex=0.8)
```




```
#text(mod.tree, use.n=TRUE, cex=0.8)
predict.tree<-predict(mod.tree,testc,type="class")
M.tree <- confusionMatrix(predict.tree,testc$classe)
M.tree
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1757   50    6    1    0
##          B   64 1030   57   48    0
##          C   20  165 1043  170   46
##          D    0    8   12  661   62
##          E    0    0   11  181 1082
##
## Overall Statistics
##
##                Accuracy : 0.8608
##                  95% CI : (0.8522, 0.8692)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.824
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9544   0.8220   0.9238   0.6230   0.9092
## Specificity            0.9877   0.9676   0.9250   0.9849   0.9637
## Pos Pred Value         0.9686   0.8590   0.7223   0.8896   0.8493
## Neg Pred Value         0.9820   0.9577   0.9829   0.9302   0.9792
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2714   0.1591   0.1611   0.1021   0.1671
## Detection Prevalence   0.2802   0.1852   0.2230   0.1148   0.1968
## Balanced Accuracy      0.9710   0.8948   0.9244   0.8039   0.9365
```

```
accuracy<-M.tree$overall[1]
plot(M.tree$table,main=paste("Decision Tree with Accurary: ", round(accuracy,4)*100, "%"))
```



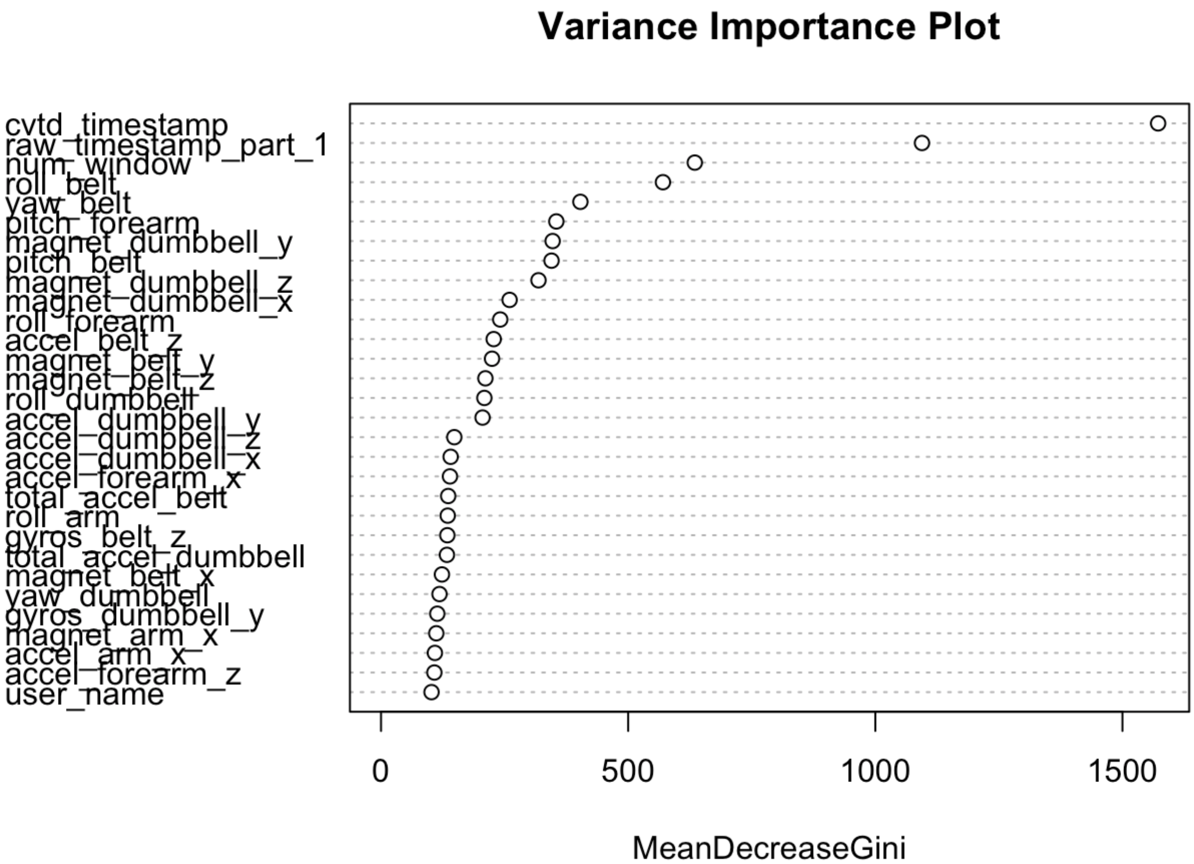
##

Random Forests RF prevents overfitting by coercing to consider a subset of variables n each split

```
set.seed(123)
sqrt(ncol(trainc)-1)
```

```
## [1] 7.549834
```

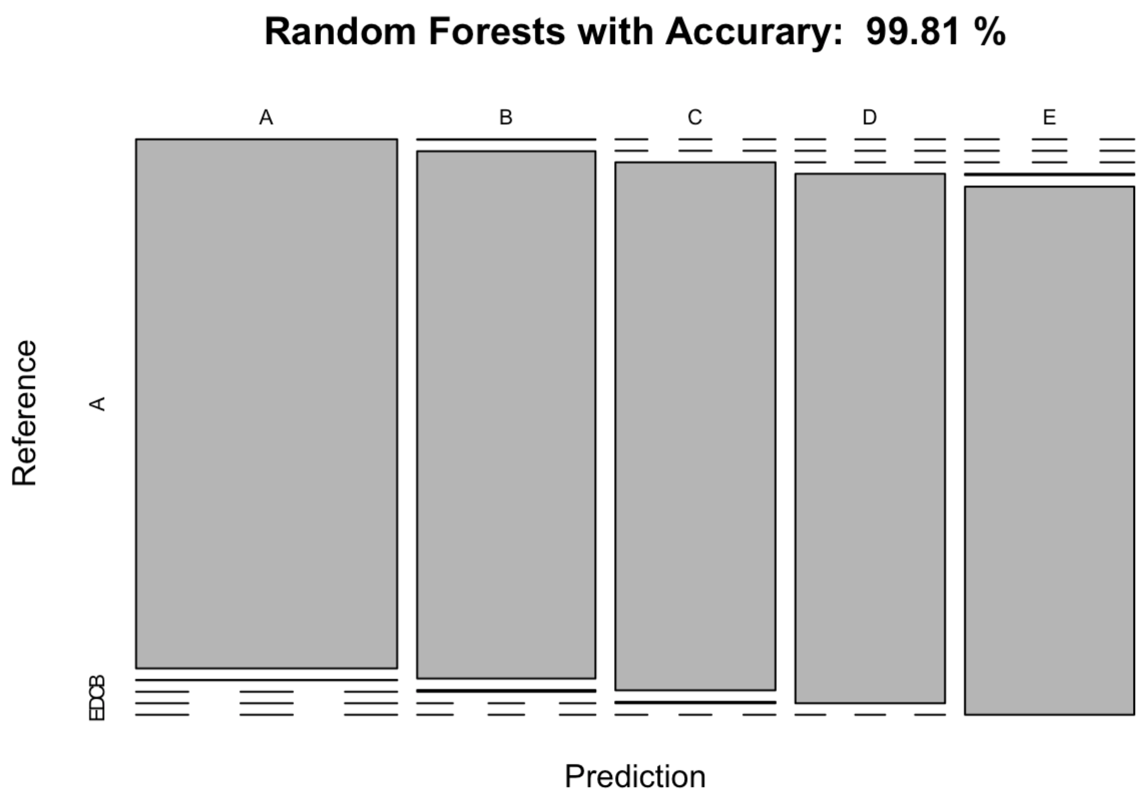
```
mod.rf <- randomForest(classe~., data=trainc, mtry=7,
                       keep.forest=TRUE)
# Usually m, number of variables considered at each split, is the square root of the total number o
f variables
varImpPlot(mod.rf,type=2,main="Variance Importance Plot")
```




```
predict.rf <- predict(mod.rf,testc,type="class")
M.rf <- confusionMatrix(predict.rf,testc$classe)
M.rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1840    1    0    0    0
##          B    1 1252    4    0    0
##          C    0    0 1125    3    0
##          D    0    0    0 1055    0
##          E    0    0    0    3 1190
##
## Overall Statistics
##
##                Accuracy : 0.9981
##                  95% CI : (0.9968, 0.999)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9977
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9995   0.9992   0.9965   0.9943   1.0000
## Specificity            0.9998   0.9990   0.9994   1.0000   0.9994
## Pos Pred Value         0.9995   0.9960   0.9973   1.0000   0.9975
## Neg Pred Value         0.9998   0.9998   0.9993   0.9989   1.0000
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2842   0.1934   0.1738   0.1630   0.1838
## Detection Prevalence   0.2844   0.1942   0.1742   0.1630   0.1843
## Balanced Accuracy      0.9996   0.9991   0.9979   0.9972   0.9997
```

```
accuracy<-M.rf$overall[1]
plot(M.rf$table,main=paste("Random Forests with Accurary: ", round(accuracy,4)*100, "%"))
```



Gradient Boosting

```
set.seed(123)
ctrl <- trainControl(number=10,repeats=5)
#The default setting performed 150 interations and took a long time; therefore, I limit the iterati
ons to 10 and use 5 fold CV
mod.gbm <- train(classe~.,data=trainc, method="gbm",
                 trControl = ctrl, verbose=FALSE)
```

```
## 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)
```

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

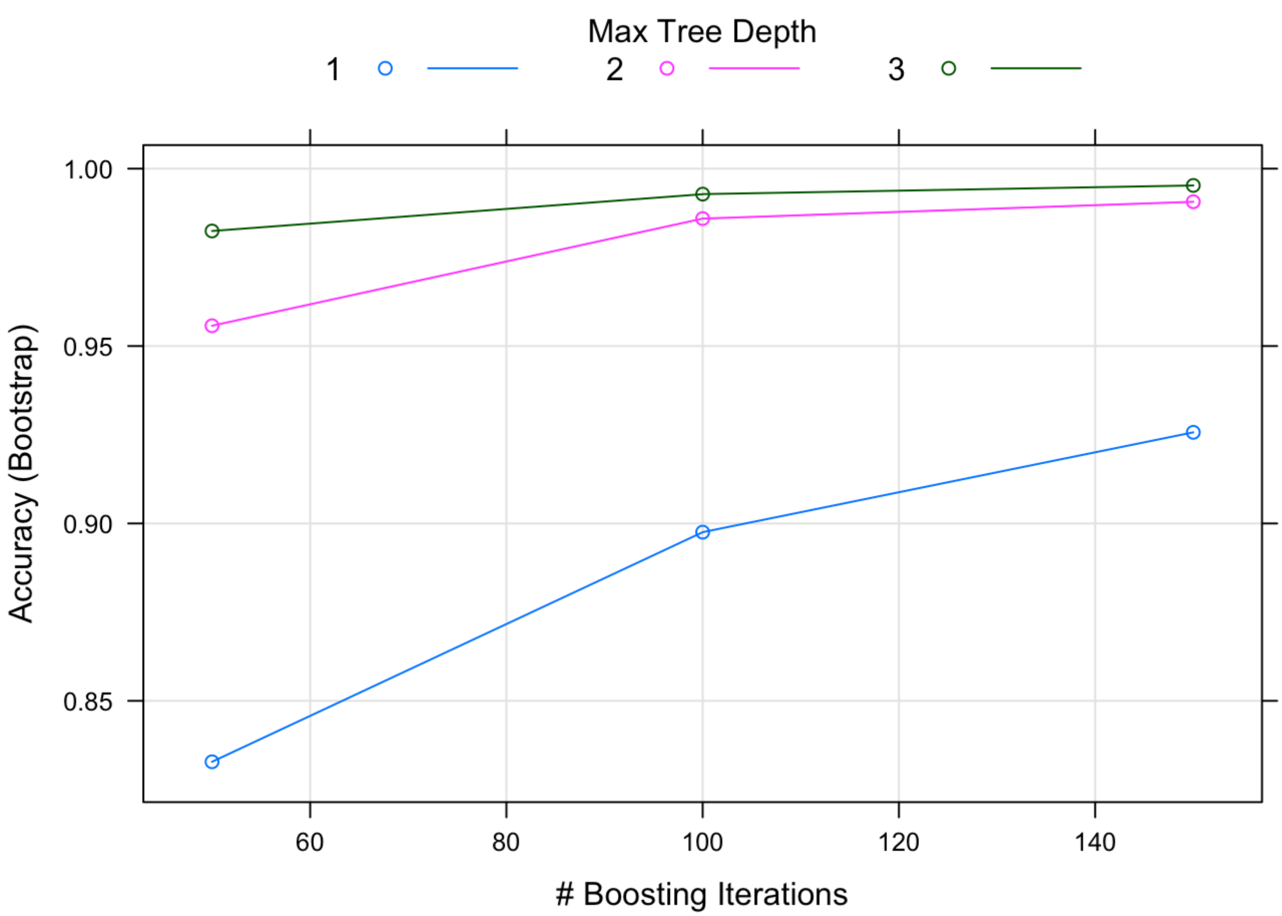
```
##
## Attaching package: 'plyr'
```

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

```
mod.gbm$finalModel
```

```
## A gradient boosted model with multinomial loss function.
## 150 iterations were performed.
## There were 79 predictors of which 45 had non-zero influence.
```

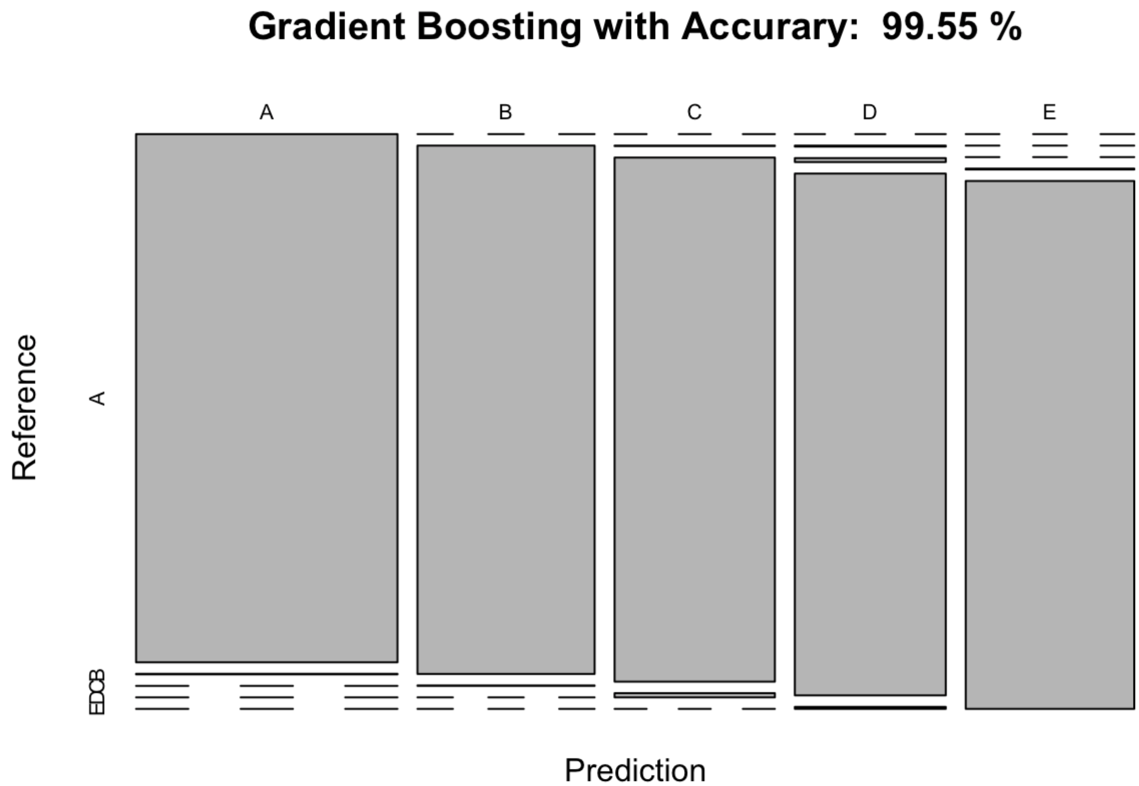
```
plot(mod.gbm)
```




```
predict.gbm <- predict(mod.gbm,testc)
M.gbm <- confusionMatrix(predict.gbm,testc$classe)
M.gbm
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1841    2    0    0    0
##          B    0 1248    1    0    0
##          C    0    1 1120    9    0
##          D    0    2    8 1050    4
##          E    0    0    0    2 1186
## 
## Overall Statistics
##                                           
##                Accuracy : 0.9955          
##                  95% CI : (0.9936, 0.997) 
##     No Information Rate : 0.2844          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.9943          
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9960   0.9920   0.9896   0.9966
## Specificity            0.9996   0.9998   0.9981   0.9974   0.9996
## Pos Pred Value         0.9989   0.9992   0.9912   0.9868   0.9983
## Neg Pred Value         1.0000   0.9990   0.9983   0.9980   0.9992
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2844   0.1928   0.1730   0.1622   0.1832
## Detection Prevalence   0.2847   0.1929   0.1745   0.1643   0.1835
## Balanced Accuracy      0.9998   0.9979   0.9951   0.9935   0.9981
```

```
accuracy<-M.gbm$overall[1]
plot(M.gbm$table,main=paste("Gradient Boosting with Accurary: ", round(accuracy,4)*100, "%"))
```



Predicting on the Test Data

Predict with Random Forests

Convert the testing set into the same type as training set

```
testing <- rbind(trainc[2, -58] , testing)
testing <- testing[-1,]
```

I choose random forest here since it is fast and has great predictor power

```
yhat <- predict(mod.rf,testing,type="class")
yhat
```

```
##  1  2  3 41  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
```

Combining models

Even though random forests, gradient boosting have high accurary on the testing set, ensembling an odd number of models usually generates the best result; however, it is time-consuming.

```
#comb <- data.frame(predict.tree,predict.rf,predict.gbm,
#                    classe=testc$classe)
#comb.mod <- train(classe~.,data=comb,method="rf")
#yhat2 <- predict(comb.mod,testing,type="class")
```