Practical Machine Learning Course Project

Nick Franciose

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).

Load the data

```
library(AppliedPredictiveModeling)
library(caret)
```

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

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

```
library(rpart)
library(rattle)
```

```
## Rattle: A free graphical interface for data mining with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

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

```
## Warning: package 'rpart.plot' was built under R version 3.2.5
```

```
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:ggplot2':
##
##     margin
```

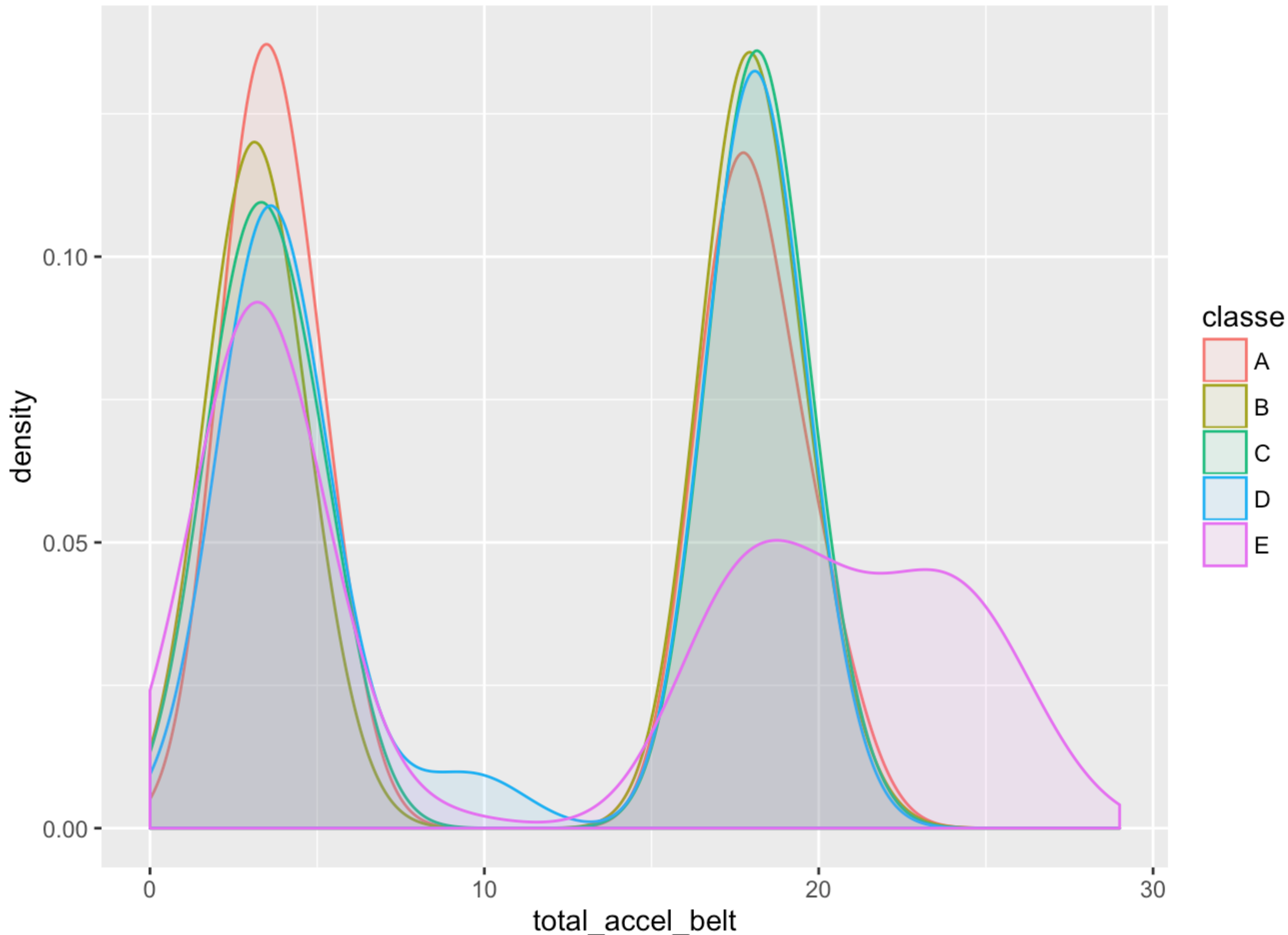
```
library(knitr)
```

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

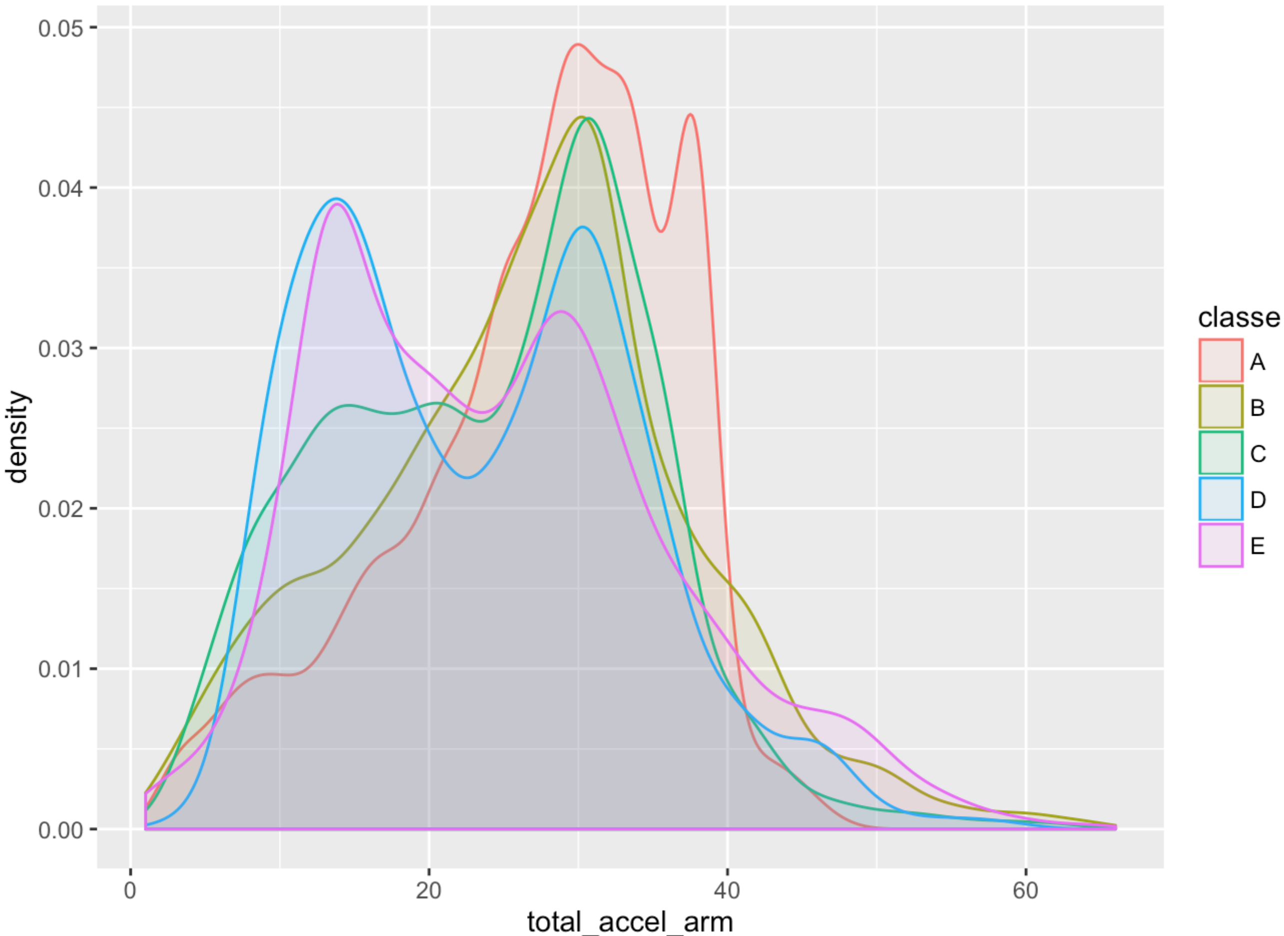
```
library(RColorBrewer)
library(ggplot2)
library(plyr)
#Read files and replace zeros with NA
train <- read.csv("pml-training.csv", na.strings=c("NA","#DIV/0!", ""), header=TRUE)
test <- read.csv("pml-testing.csv", na.strings=c("NA","#DIV/0!", ""), header=TRUE)
```

Clean the data

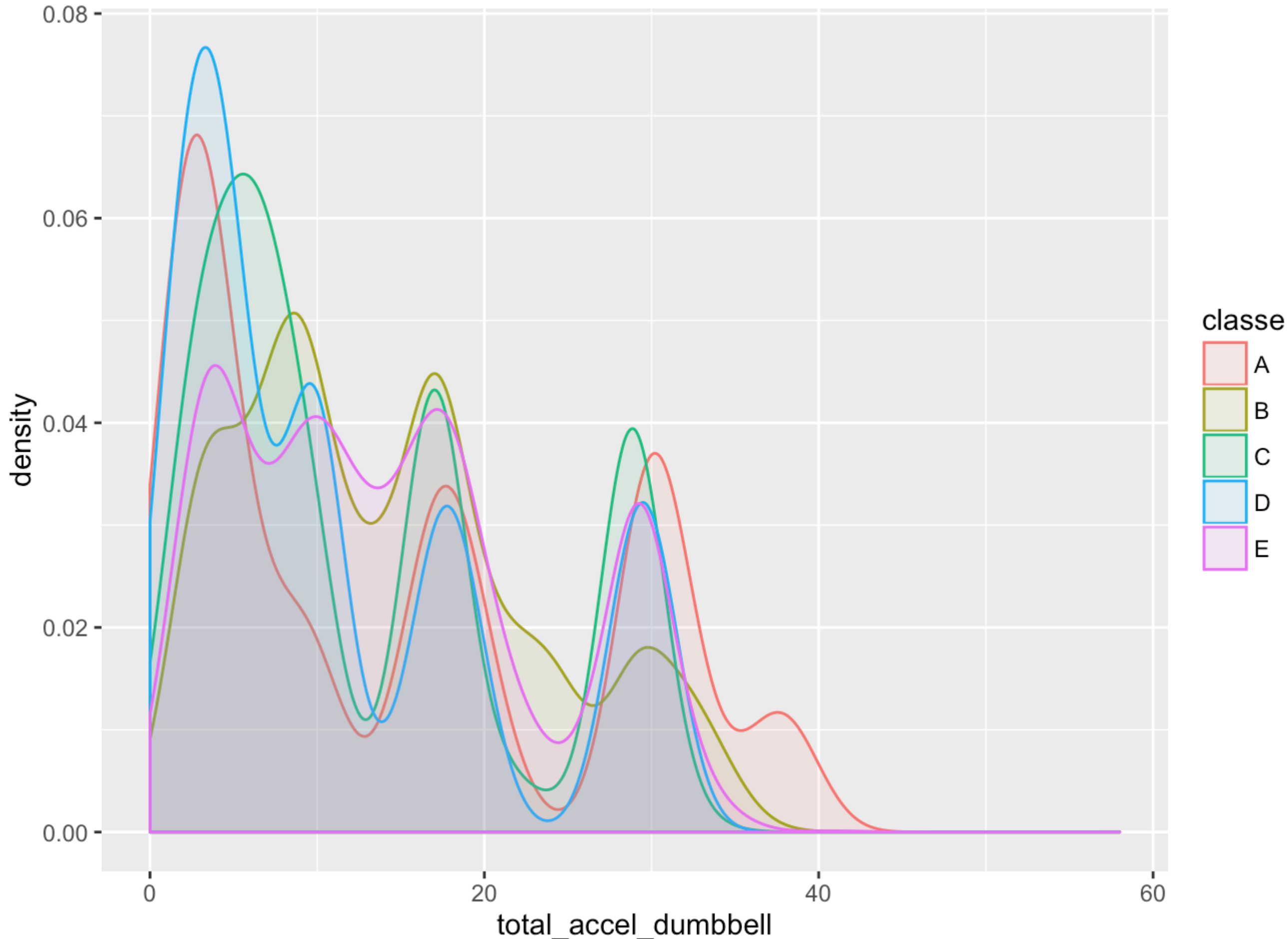
```
#Remove the first seven columns
train <- train[-c(1:7)]

#Remove NearZeroVariance variables

#nearZeroVar <- nearZeroVar(train, saveMetrics=TRUE)
#train <- train[!nearZeroVar]


#Clean variables with more than 70% NA

cleanedtrain<- train
for(i in 1:length(train)) {
    if( sum( is.na( train[, i] ) ) /nrow(train) >= .7) {
        for(j in 1:length(cleanedtrain)) {
            if( length( grep(names(train[i]), names(cleanedtrain)[j]) ) == 1)  {
                cleanedtrain <- cleanedtrain[ , -j]
            }
        }
    }
}

# Set back to the original variable name
train <- cleanedtrain
rm(cleanedtrain); rm(i); rm(j)

#Explore the Test Data
ggplot(train, aes(total_accel_belt, colour = classe, fill = classe)) +geom_density(alpha
 = 0.1)
```



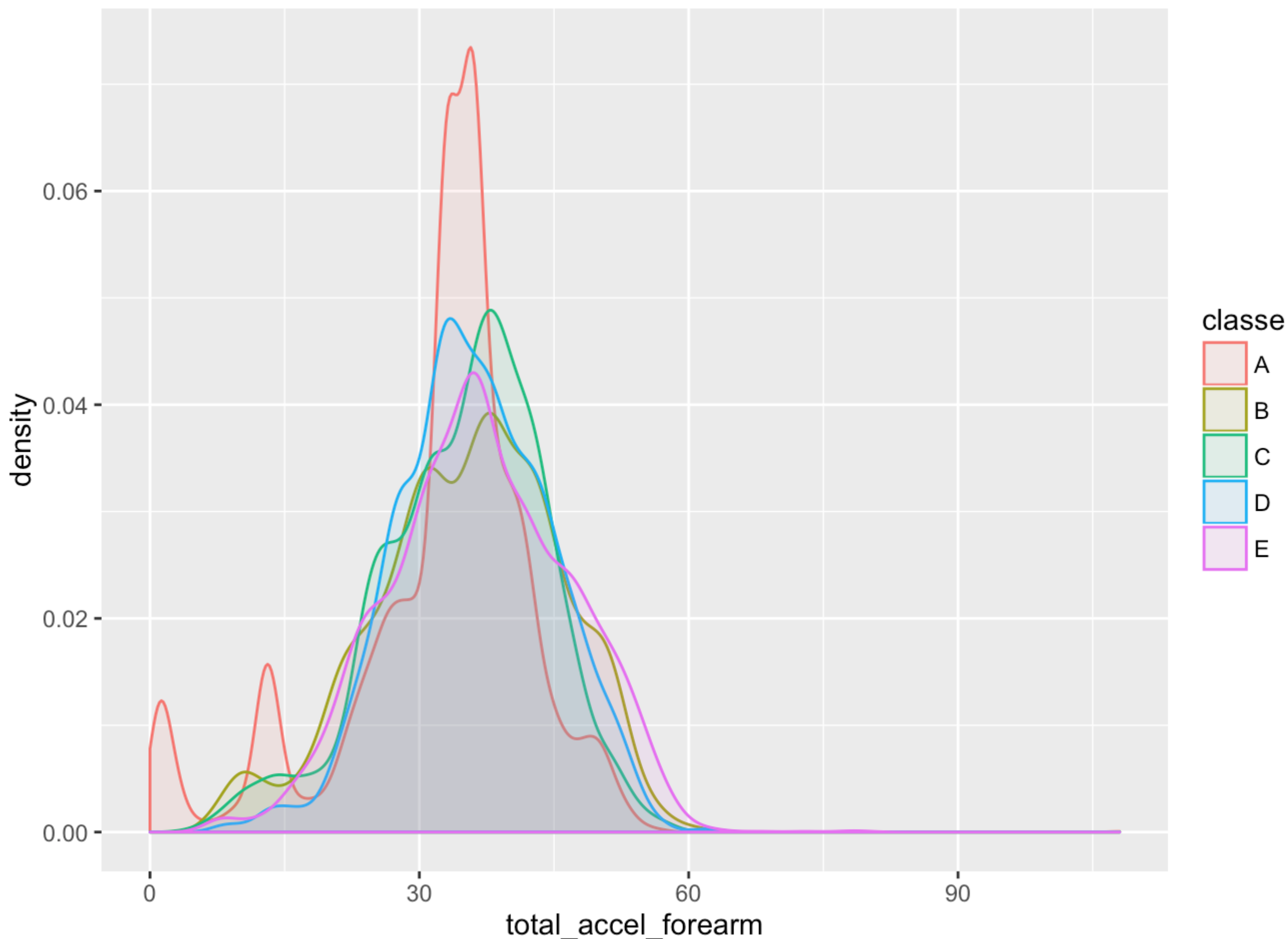

```
ggplot(train, aes(total_accel_arm, colour = classe, fill = classe)) +geom_density(alpha
= 0.1)
```




```
ggplot(train, aes(total_accel_dumbbell, colour = classe, fill = classe)) +geom_density(a
lpha = 0.1)
```




```
ggplot(train, aes(total_accel_forearm, colour = classe, fill = classe)) +geom_density(al
pha = 0.1)
```




```
#Partioning the training set into two
trainPart <- createDataPartition(train$classe, p=0.6, list=FALSE)
myTrain <- train[trainPart, ]
myTest <- train[-trainPart, ]
dim(myTrain); dim(myTest)
```

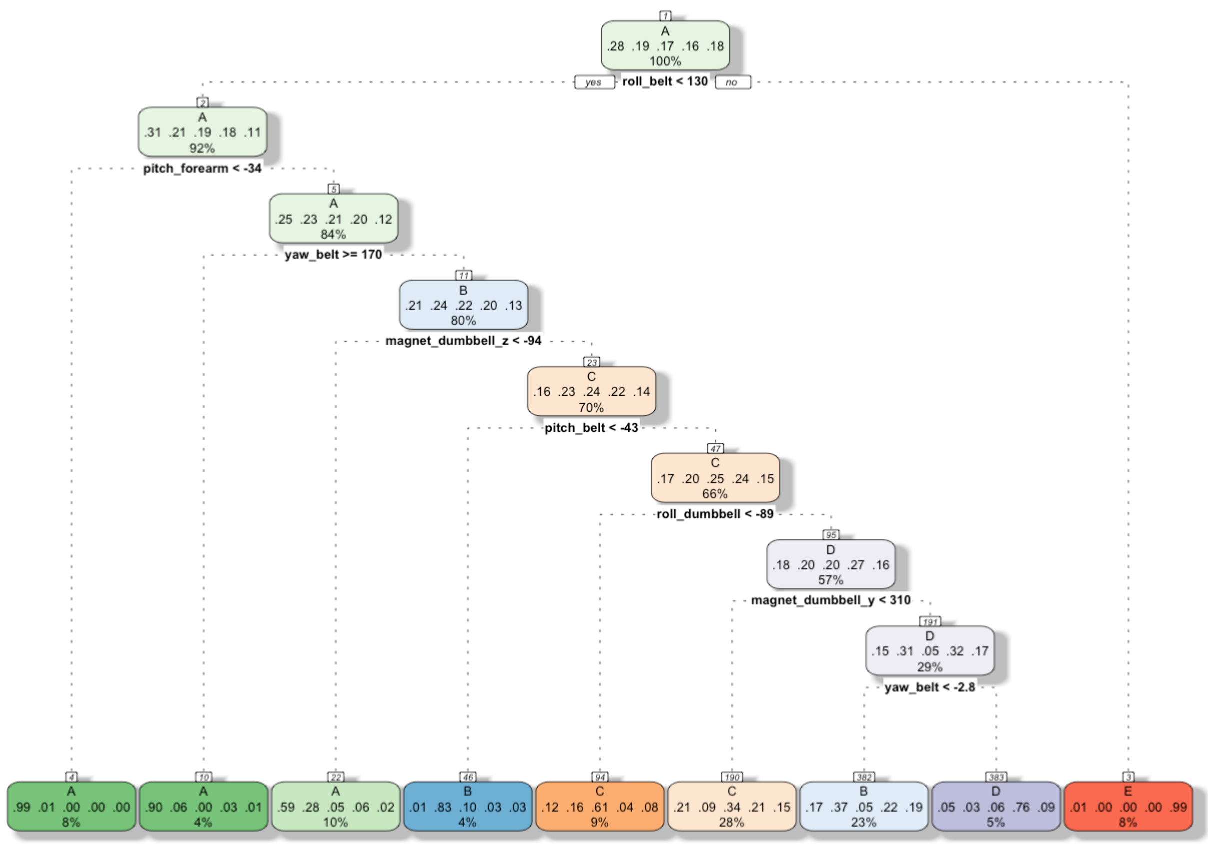
```
## [1] 11776    53
```

```
## [1] 7846   53
```

```
#Classification Tree

set.seed(666)
ctFit <- train(myTrain$classe ~ ., data = myTrain, method="rpart")
print(ctFit, digits=3)
```

```
## CART
##
## 11776 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 11776, 11776, 11776, 11776, 11776, 11776, ...
## Resampling results across tuning parameters:
##
##   cp      Accuracy  Kappa   Accuracy SD  Kappa SD
##   0.0218  0.590     0.4770  0.0392       0.0559
##   0.0270  0.559     0.4374  0.0255       0.0377
##   0.1159  0.334     0.0745  0.0413       0.0622
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.0218.
```

```
fancyRpartPlot(ctFit$finalModel)
```



Rattle 2016-Oct-21 16:57:35 nickfranciose

```
#Apply Classification tree to myTest
ctPredictions <- predict(ctFit, newdata=myTest)
print(confusionMatrix(ctPredictions, myTest$classe), digits=4)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1372  222   36   70   23
##          B  320  958  123  407  374
##          C  528  320 1180  483  361
##          D    7   18   29  326   39
##          E    5    0    0    0  645
##
## Overall Statistics
##
##                Accuracy : 0.5711
##                  95% CI : (0.5601, 0.5821)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4612
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.6147   0.6311   0.8626  0.25350  0.44730
## Specificity            0.9375   0.8066   0.7388  0.98582  0.99922
## Pos Pred Value         0.7963   0.4390   0.4109  0.77804  0.99231
## Neg Pred Value         0.8595   0.9011   0.9622  0.87074  0.88924
## Prevalence             0.2845   0.1935   0.1744  0.16391  0.18379
## Detection Rate         0.1749   0.1221   0.1504  0.04155  0.08221
## Detection Prevalence   0.2196   0.2781   0.3660  0.05340  0.08284
## Balanced Accuracy      0.7761   0.7188   0.8007  0.61966  0.72326
```

```
#Create a prediction model using boosting with Trees

#TRAIN
gbmFit <- train(classe ~ ., data = train, method = "gbm", 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
```

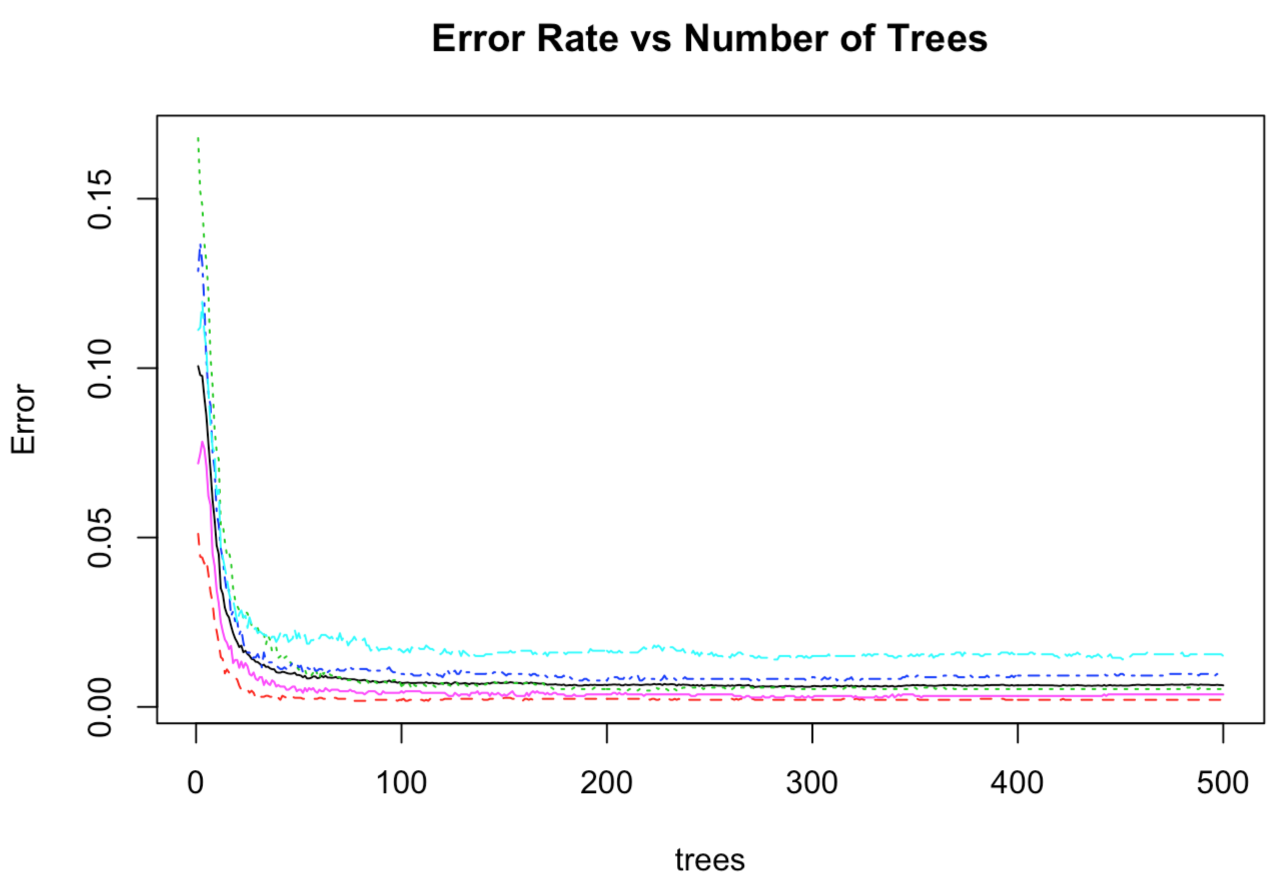
```
#Test Boosting with Trees model
gbmPrediction <- predict(gbmFit, myTest)
confusionMatrix(gbmPrediction, myTest$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2212   38    0    1    4
##          B   14 1449   33    3    9
##          C    4   30 1325   30    8
##          D    0    1    8 1246   11
##          E    2    0    2    6 1410
##
## Overall Statistics
##
##                Accuracy : 0.974
##                  95% CI : (0.9702, 0.9774)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9671
##  Mcnemar's Test P-Value : 2.469e-06
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9910   0.9545   0.9686   0.9689   0.9778
## Specificity            0.9923   0.9907   0.9889   0.9970   0.9984
## Pos Pred Value         0.9809   0.9609   0.9485   0.9842   0.9930
## Neg Pred Value         0.9964   0.9891   0.9933   0.9939   0.9950
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2819   0.1847   0.1689   0.1588   0.1797
## Detection Prevalence   0.2874   0.1922   0.1781   0.1614   0.1810
## Balanced Accuracy      0.9917   0.9726   0.9787   0.9829   0.9881
```

```
#Create Random Forest model
set.seed(1777)
random_forest=randomForest(classe~.,data=myTrain,ntree=500,importance=TRUE)
random_forest
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = myTrain, ntree = 500,      importance = TR
UE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.64%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3341    5    0    0    2 0.002090800
## B    8 2267    4    0    0 0.005265467
## C    0   17 2035    2    0 0.009250243
## D    0    0   27 1901    2 0.015025907
## E    0    0    1    7 2157 0.003695150
```

```
plot(random_forest,main="Error Rate vs Number of Trees")
```




```
rfPredictions = predict(random_forest, newdata=myTest)
confusionMatrix(rfPredictions,myTest$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2229    9    0    0    0
##          B    3 1506   10    0    0
##          C    0    3 1357   19    0
##          D    0    0    1 1267    0
##          E    0    0    0    0 1442
##
## Overall Statistics
##
##                Accuracy : 0.9943
##                  95% CI : (0.9923, 0.9958)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9927
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9987   0.9921   0.9920   0.9852   1.0000
## Specificity            0.9984   0.9979   0.9966   0.9998   1.0000
## Pos Pred Value         0.9960   0.9914   0.9840   0.9992   1.0000
## Neg Pred Value         0.9995   0.9981   0.9983   0.9971   1.0000
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2841   0.1919   0.1730   0.1615   0.1838
## Detection Prevalence   0.2852   0.1936   0.1758   0.1616   0.1838
## Balanced Accuracy      0.9985   0.9950   0.9943   0.9925   1.0000
```

```
#Explore Variable importance to Random Forest model
```

Applying RF model to test set

The random forest is the most accurate, so we will apply this model to the test set.

```
#Remove all unneeded columns
test <-test[,c(8:11,37:49,60:68,84:86,102,113:124,140,151:160)]
print(predict(random_forest, newdata=test))
```

```
##  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
```