Coursera Practical Machine Learning Course Proj

John Kerwin Ty

August 1, 2019

We first load the libraries needed

```
library(dplyr);library(caret);library(rpart)
```

```
##
## 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
```

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

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

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
library(rpart.plot);library(rattle);library(randomForest);library(corrplot)
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
## Warning: package 'randomForest' was built under R version 3.6.1
```

```
## 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
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## Warning: package 'corrplot' was built under R version 3.6.1
```

```
## corrplot 0.84 loaded
```

```
train<-read.csv("pml-training.csv");test<-read.csv("pml-testing.csv")
```

Data Pre-processing

We create a data partition with the training dataset and split it 70% train and 30% test.

```
# Set seed for reproducible
set.seed(12345)
inTrain  <- createDataPartition(train$classe, p=0.7, list=FALSE)
TrainSet <- train[inTrain, ];TestSet  <- train[-inTrain, ]
dim(TrainSet);dim(TestSet)
```

```
## [1] 13737   160
```

```
## [1] 5885  160
```

```
# We want to see an overview of the contents for each variable
str(TrainSet)
```

```
## 'data.frame':    13737 obs. of  160 variables:
##  $ X                       : int  1 2 3 5 9 10 13 14 15 16 ...
##  $ user_name               : Factor w/ 6 levels "adelmo","carlitos",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ raw_timestamp_part_1    : int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 1323084232 1323
084232 1323084232 1323084232 ...
##  $ raw_timestamp_part_2    : int  788290 808298 820366 196328 484323 484434 560359 576390 604281 644302 ...
##  $ cvtd_timestamp          : Factor w/ 20 levels "02/12/2011 13:32",..: 9 9 9 9 9 9 9 9 9 9 ...
##  $ new_window              : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ num_window              : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ roll_belt               : num  1.41 1.41 1.42 1.48 1.43 1.45 1.42 1.42 1.45 1.48 ...
##  $ pitch_belt              : num  8.07 8.07 8.07 8.07 8.16 8.17 8.2 8.21 8.2 8.15 ...
##  $ 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 ...
##  $ kurtosis_roll_belt      : Factor w/ 397 levels "","-0.016850",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_belt     : Factor w/ 317 levels "","-0.021887",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt      : Factor w/ 395 levels "","-0.003095",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt.1    : Factor w/ 338 levels "","-0.005928",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_belt            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ min_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_belt            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ amplitude_roll_belt     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_belt    : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_belt      : Factor w/ 4 levels "","#DIV/0!","0.00",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ var_total_accel_belt    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_belt        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_belt       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_belt         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_belt_x            : num  0 0.02 0 0.02 0.02 0.03 0.02 0.02 0 0 ...
##  $ gyros_belt_y            : num  0 0 0 0.02 0 0 0 0 0 0 ...
##  $ gyros_belt_z            : num  -0.02 -0.02 -0.02 -0.02 -0.02 0 0 -0.02 0 0 ...
##  $ accel_belt_x            : int  -21 -22 -20 -21 -20 -21 -22 -22 -21 -21 ...
##  $ accel_belt_y            : int  4 4 5 2 2 4 4 4 2 4 ...
##  $ accel_belt_z            : int  22 22 23 24 24 22 21 21 22 23 ...
##  $ magnet_belt_x           : int  -3 -7 -2 -6 1 -3 -3 -8 -1 0 ...
##  $ magnet_belt_y           : int  599 608 600 600 602 609 606 598 597 592 ...
##  $ magnet_belt_z           : int  -313 -311 -305 -302 -312 -308 -309 -310 -310 -305 ...
##  $ roll_arm                : num  -128 -128 -128 -128 -128 -128 -128 -128 -129 -129 ...
##  $ pitch_arm               : num  22.5 22.5 22.5 22.1 21.7 21.6 21.4 21.4 21.4 21.3 ...
##  $ 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 ...
##  $ var_accel_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_arm         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_arm        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_arm_x             : num  0 0.02 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 ...
##  $ gyros_arm_y             : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.02 0 0 0 ...
##  $ gyros_arm_z             : num  -0.02 -0.02 -0.02 0 -0.02 -0.02 -0.02 -0.03 -0.03 -0.03 ...
```

```
##  $ accel_arm_x            : int  -288 -290 -289 -289 -288 -288 -287 -288 -289 -289 ...
##  $ accel_arm_y            : int  109 110 110 111 109 110 111 111 111 109 ...
##  $ accel_arm_z            : int  -123 -125 -126 -123 -122 -124 -124 -124 -124 -121 ...
##  $ magnet_arm_x           : int  -368 -369 -368 -374 -369 -376 -372 -371 -374 -367 ...
##  $ magnet_arm_y           : int  337 337 344 337 341 334 338 331 342 340 ...
##  $ magnet_arm_z           : int  516 513 513 506 518 516 509 523 510 509 ...
##  $ kurtosis_roll_arm      : Factor w/ 330 levels "","-0.02438",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_arm     : Factor w/ 328 levels "","-0.00484",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_arm       : Factor w/ 395 levels "","-0.01548",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_arm      : Factor w/ 331 levels "","-0.00051",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_pitch_arm     : Factor w/ 328 levels "","-0.00184",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_arm       : Factor w/ 395 levels "","-0.00311",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_arm            : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_roll_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_arm            : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_arm     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_arm    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_arm      : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ roll_dumbbell          : num  13.1 13.1 12.9 13.4 13.2 ...
##  $ 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 ...
##  $ kurtosis_roll_dumbbell  : Factor w/ 398 levels "","-0.0035","-0.0073",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_dumbbell : Factor w/ 401 levels "","-0.0163","-0.0233",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_dumbbell   : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_dumbbell  : Factor w/ 401 levels "","-0.0082","-0.0096",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_pitch_dumbbell : Factor w/ 402 levels "","-0.0053","-0.0084",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_dumbbell   : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_dumbbell        : Factor w/ 73 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ min_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_dumbbell        : Factor w/ 73 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ amplitude_roll_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##   [list output truncated]
```

We have to first clean the data since if you can notice that most of the columns have NA values or no values at all. We want to remove thsoe columns since they don't provide significant information for our model.

```
# Here we get the indexes of the columns having at least 90% of NA or blank values on the training dataset

dropcol <- which(colSums(is.na(TrainSet) |TrainSet=="")>0.9*dim(TrainSet)[1])
TrainSet_Clean <- TrainSet[,-dropcol]

# We also remove the first 7 columns since it doesn't give any valuable information
TrainSet_Clean <- TrainSet_Clean[,-c(1:7)]
dim(TrainSet_Clean)
```

```
## [1] 13737    53
```

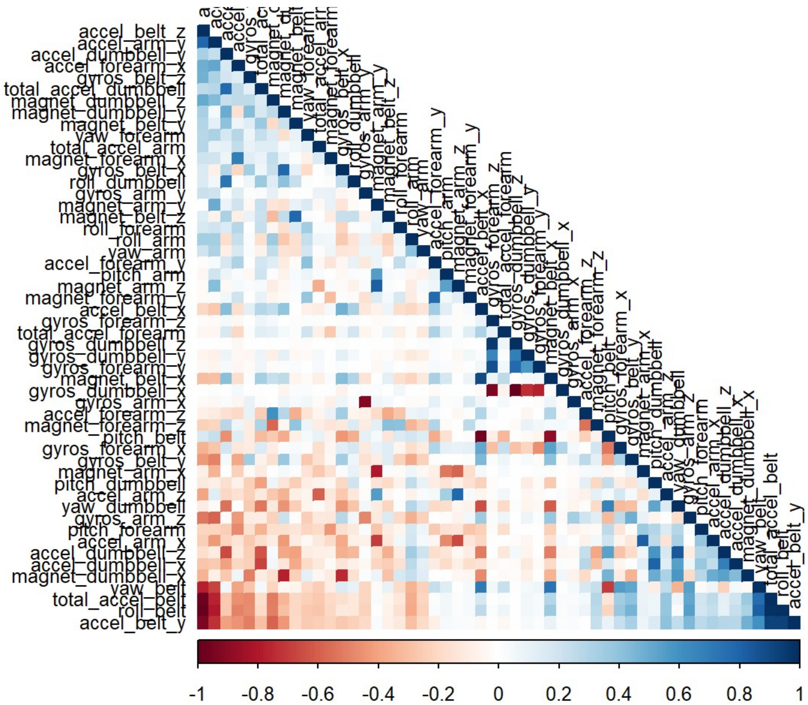
```
# We also do the same thing with the test data
TestSet_Clean <- TestSet[,-dropcol]
TestSet_Clean <- TestSet_Clean[,-c(1:7)]
dim(TestSet_Clean)
```

```
## [1] 5885   53
```

Exploratory Analysis on Data

We also want to analyze the correlation between variables; hence we plot using a correlation matrix where the highly correlated variables are shown in dark color.

```
corMatrix <- cor(TrainSet_Clean[,-53])
corrplot(corMatrix, order = "FPC", method = "color", type = "lower",
         tl.cex = 0.8, tl.col = rgb(0, 0, 0))
```



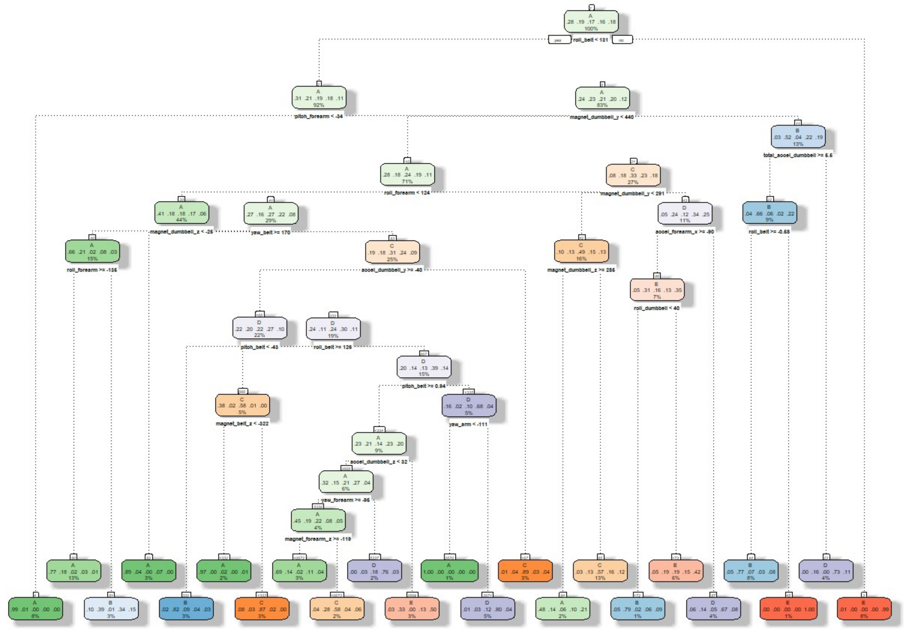
To make an evem more compact analysis, a PCA (Principal Components Analysis) could be performed as pre-processing step to the datasets given we sacrifice the interpretability of the model. Nevertheless, as the correlations are quite few, this step will not be applied for this assignment.

Predictive Modeling

We train our model with the training set and then we use the test set to see the performance of each model. In this report, we are going to train with 1) Decision Trees 2) Random Forest and 3) Gradient Boosted Model. Afterwards, we are going to choose the best model according to the accuracy of each model. Furthermore, a confusion matrix is generated to further explain each model.

Method 1 : Decision Trees

```
set.seed(12345)
modFitDecTree <- rpart(classe ~ ., data=TrainSet_Clean, method="class")
fancyRpartPlot(modFitDecTree)
```



Rattle 2019-Aug-03 17:59:07 10012220

```
# prediction on Test dataset
predictDecTree <- predict(modFitDecTree, newdata=TestSet_Clean, type="class")
confMatDecTree <- confusionMatrix(predictDecTree, TestSet_Clean$classe)
confMatDecTree
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1532  176   28   48   41
##          B   54  585   57   64   76
##          C   35  154  819  134  126
##          D   25   76   58  631   56
##          E   28  148   64   87  783
## 
## Overall Statistics
##                                          
##                Accuracy : 0.7392         
##                  95% CI : (0.7277, 0.7503)
##     No Information Rate : 0.2845         
##     P-Value [Acc > NIR] : < 2.2e-16      
##                                          
##                   Kappa : 0.6692         
##                                          
##  Mcnemar's Test P-Value : < 2.2e-16      
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9152  0.51361   0.7982   0.6546   0.7237
## Specificity            0.9304  0.94711   0.9076   0.9563   0.9319
## Pos Pred Value         0.8395  0.69976   0.6459   0.7459   0.7054
## Neg Pred Value         0.9650  0.89028   0.9552   0.9339   0.9374
## Prevalence             0.2845  0.19354   0.1743   0.1638   0.1839
## Detection Rate         0.2603  0.09941   0.1392   0.1072   0.1331
## Detection Prevalence   0.3101  0.14206   0.2155   0.1438   0.1886
## Balanced Accuracy      0.9228  0.73036   0.8529   0.8054   0.8278
```

Method 2 : Random Forest

```
# model fit
set.seed(12345)
controlRF <- trainControl(method="cv", number=3, verboseIter=FALSE)
modFitRandForest <- train(classe ~ ., data=TrainSet_Clean, method="rf",
                          trControl=controlRF)
modFitRandForest$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 0.68%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3899    5    0    0    2 0.001792115
## B   19 2630    9    0    0 0.010534236
## C    0   15 2373    8    0 0.009599332
## D    0    1   21 2227    3 0.011101243
## E    0    3    4    3 2515 0.003960396
```

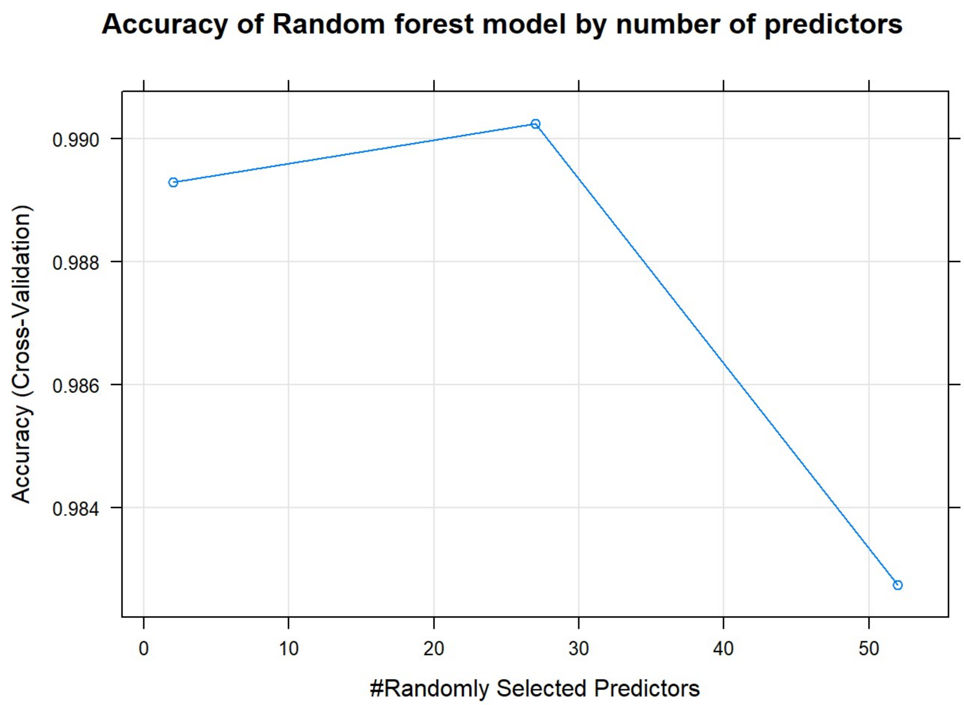
To obtain the confusion matrix and also the accuracy;

```
predictRandForest <- predict(modFitRandForest, newdata=TestSet_Clean)
confMatRandForest <- confusionMatrix(predictRandForest, TestSet_Clean$classe)
confMatRandForest
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1672    7    0    0    0
##          B    1 1129    4    0    0
##          C    1    3 1019    7    1
##          D    0    0    3  956    1
##          E    0    0    0    1 1080
## 
## Overall Statistics
##                                           
##                Accuracy : 0.9951          
##                  95% CI : (0.9929, 0.9967)
##     No Information Rate : 0.2845          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.9938          
##                                           
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9988   0.9912   0.9932   0.9917   0.9982
## Specificity            0.9983   0.9989   0.9975   0.9992   0.9998
## Pos Pred Value         0.9958   0.9956   0.9884   0.9958   0.9991
## Neg Pred Value         0.9995   0.9979   0.9986   0.9984   0.9996
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2841   0.1918   0.1732   0.1624   0.1835
## Detection Prevalence   0.2853   0.1927   0.1752   0.1631   0.1837
## Balanced Accuracy      0.9986   0.9951   0.9954   0.9954   0.9990
```

Evidently, the random forest has a high accuracy with 99.51%. So far, this is far better than method 1 (Decision Trees). We want to dig deeper on this model so we plot what is the optimal and minimum number of variables for this model. With this, we can further minimize the number of variables to be used for training in the future. Also, using the r varImp to determine what are the most important features for this model.

```
plot(modFitRandForest,main="Accuracy of Random forest model by number of predictors")
```




```
MostImpVars <- varImp(modFitRandForest)
MostImpVars
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 52)
##
##                      Overall
## roll_belt            100.000
## pitch_forearm         59.554
## yaw_belt              55.749
## magnet_dumbbell_y     46.016
## pitch_belt            44.390
## roll_forearm          43.382
## magnet_dumbbell_z     43.057
## accel_dumbbell_y      22.253
## accel_forearm_x       18.336
## magnet_dumbbell_x     16.190
## roll_dumbbell         15.782
## magnet_belt_z         15.712
## accel_belt_z          14.499
## magnet_forearm_z      14.409
## accel_dumbbell_z      13.468
## total_accel_dumbbell  12.175
## magnet_belt_y         12.075
## gyros_belt_z          10.982
## yaw_arm               10.595
## magnet_belt_x          9.608
```

Method 3 : Gradient Boosted Model

```
# model fit
set.seed(12345)
controlGBM <- trainControl(method = "repeatedcv", number = 5, repeats = 1)
modFitGBM  <- train(classe ~ ., data=TrainSet_Clean, method = "gbm",
                    trControl = controlGBM, verbose = FALSE)
modFitGBM$finalModel
```

```
## A gradient boosted model with multinomial loss function.
## 150 iterations were performed.
## There were 52 predictors of which 51 had non-zero influence.
```

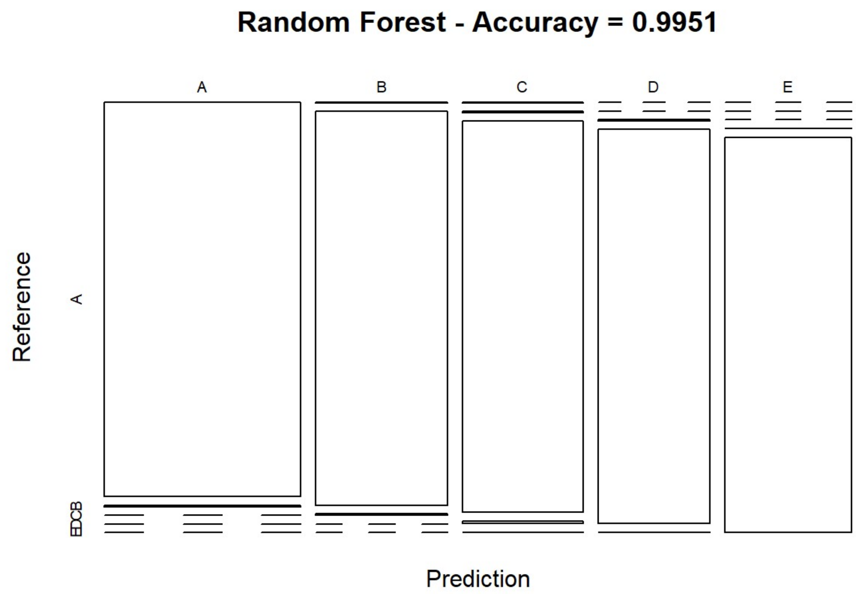
```
# prediction on Test dataset
predictGBM <- predict(modFitGBM, newdata=TestSet_Clean)
confMatGBM <- confusionMatrix(predictGBM, TestSet_Clean$classe)
confMatGBM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1647   39    0    1    1
##          B   19 1066   38    4   14
##          C    4   33  979   38    6
##          D    4    0    8  915    8
##          E    0    1    1    6 1053
##
## Overall Statistics
##
##                Accuracy : 0.9618
##                  95% CI : (0.9565, 0.9665)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9516
##
##  Mcnemar's Test P-Value : 8.329e-08
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9839   0.9359   0.9542   0.9492   0.9732
## Specificity            0.9903   0.9842   0.9833   0.9959   0.9983
## Pos Pred Value         0.9757   0.9343   0.9236   0.9786   0.9925
## Neg Pred Value         0.9936   0.9846   0.9903   0.9901   0.9940
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2799   0.1811   0.1664   0.1555   0.1789
## Detection Prevalence   0.2868   0.1939   0.1801   0.1589   0.1803
## Balanced Accuracy      0.9871   0.9601   0.9688   0.9726   0.9858
```

Conclusion and Recommendations

In conclusion, the best model is the random forest.I would suggest reducing the number of variables again in the future to make the processing faster also.

```
# plot matrix results
plot(confMatRandForest$table, col = confMatRandForest$byClass,
     main = paste("Random Forest - Accuracy =",
                  round(confMatRandForest$overall['Accuracy'], 4)))
```



With this in mind, we now predict the test (or rather the valid) set for our 20 items exam as part of the final requirements for this module.

```
predictVALID <- predict(modFitRandForest, newdata=test)
predictVALID
```

```
##  [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
```