PRACTICAL MACHINE LEARNING PROJECT: "PREDICTING HUMAN EXERCISE USING SELF-MONITORING DEVICES"

Pilar Cantero

19 de septiembre de 2016

EXECUTIVE SUMMARY

The emergence of the digital age has been impacted with several technological changes where people are willing to measure their own individual daily activities related to work, exercise, sleep, diet, mood, etc. We are covering ourselves up with these new "gadgets", such as Fitbit, Jawbone Up and Nike FuelBand, which are collecting all this information. These type of devices are part of the quantified self movement and 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 our analysis we carried out an experiment with a group of 6 participants (aged between 20-28 years) using data from accelerometers on the belt, forearm, arm and dumbbell, to build a model to predict the manner in which these participants did the exercise, and then to predict the movement of 20 different test cases.

They were asked to perform barbell lifts correctly and incorrectly in 5 different ways: exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E). Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes.

More information is available from the website here: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Thus, it is an interesting problem to build a model that predicts what kind of exercise a subject is performing based on the quantitative measurements from self monitoring devices.

Our analysis suggests that our prediction function, developed using the Random Forests method, will have a great accuracy (over 99.70%) to predict the 20 test cases with 100% accuracy.

BASIC SETTING

RStudio

knitr

echo = TRUE

set.seed(12345)

Load libraries:

library(caret) library(rpart) library(rpart.plot) library(rattle) library(RColorBrewer) library(randomForest) library("e1071") library(gbm) library(ggplot2) library (gridExtra).

GETTING AND CLEANING DATA

The data for this project come from this original source: http://groupware.les.inf.puc-rio.br/har

Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013.

I would like to thank the authors for being very generous in allowing their data to be used for this kind of assignment.

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

The test data are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

Downloading and reading the data:

```
trainurl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
```

```
traindata <- read.csv(url(trainurl), na.strings=c("NA","#DIV/
0!",""))
dim(traindata)
## [1] 19622   160
testurl <- "http://d396qusza40orc.cloudfront.net/
predmachlearn/pml-testing.csv"
validationdata <- read.csv(url(testurl), na.strings=c
("NA","#DIV/0!",""))
dim(validationdata)
## [1]  20 160
```

Getting a subtraining data set and a subtesting data set from the original training data set to be used for Cross Validation:

Dividing the original traindata set into two subdata sets: 60% in the finaltrain data set and 40% in the finaltest data set. I will perform cross validation within the training division in order to improve the model fit. After that, I will do out-of-sample test with the testing division to validate the model where an expected out-of-sample error rate of less than 0.5%, or 99.5% accuracy, would be acceptable before it is used to perform the prediction on the 20 test cases (that must have 100% accuracy to obtain 20 points awarded). Therefore, I leave the original test set (validationdata) alone, and I will apply our ultimate prediction algorithm to this test set in order to be an unbiased measurement.

```
library (caret)
## Loading required package: lattice
## Loading required package: ggplot2
subtrain <- createDataPartition(traindata$classe, p=0.6,
list=FALSE)
finaltrain <- traindata[subtrain, ]
finaltest<- traindata[-subtrain, ]
dim(finaltrain)
## [1] 11776   160
dim(finaltest)
## [1] 7846  160
```

Cleaning data

I will have a look at these subdata sets and I will call the nearZeroVar function with the argument saveMetrics = TRUE

```
x = nearZeroVar(finaltrain, saveMetrics = TRUE)
 str(x, vec.len=2)
## 'data.frame':    160 obs. of  4 variables:
##  $ freqRatio    : num  1 1.1 ...
##  $ percentUnique: num  100 0.051 ...
##  $ zeroVar      : logi  FALSE FALSE FALSE ...
##  $ nzv          : logi  FALSE FALSE FALSE ...
y = nearZeroVar(finaltest, saveMetrics = TRUE)
str(y, vec.len=2)
## 'data.frame':    160 obs. of  4 variables:
##  $ freqRatio    : num  1 1.1 ...
##  $ percentUnique: num  100 0.0765 ...
##  $ zeroVar      : logi  FALSE FALSE FALSE ...
##  $ nzv          : logi  FALSE FALSE FALSE ...
```

By default, a predictor is classified as near-zero variance if the percentage of unique values in the samples is less than 10% and when the frequency ratio mentioned above is greater than 19 (95/5).

We can explore which ones are the zero variance predictors:

```
x[x[,"zeroVar"] > 0, ]
##                          freqRatio percentUnique zeroVar
nzv
## kurtosis_yaw_belt                0   0.000000000    TRUE
TRUE
## skewness_yaw_belt                0   0.000000000    TRUE
TRUE
## amplitude_yaw_belt               0   0.008491848    TRUE
TRUE
## kurtosis_yaw_dumbbell            0   0.000000000    TRUE
TRUE
## skewness_yaw_dumbbell            0   0.000000000    TRUE
TRUE
## amplitude_yaw_dumbbell           0   0.008491848    TRUE
TRUE
## kurtosis_yaw_forearm             0   0.000000000    TRUE
TRUE
## skewness_yaw_forearm             0   0.000000000    TRUE
TRUE
## amplitude_yaw_forearm            0   0.008491848    TRUE
TRUE
y[y[,"zeroVar"] > 0, ]
##                          freqRatio percentUnique zeroVar
nzv
## kurtosis_yaw_belt                0    0.00000000    TRUE
TRUE
```

```
## skewness_yaw_belt                 0     0.00000000     TRUE
TRUE
## amplitude_yaw_belt                0     0.01274535     TRUE
TRUE
## kurtosis_yaw_dumbbell             0     0.00000000     TRUE
TRUE
## skewness_yaw_dumbbell             0     0.00000000     TRUE
TRUE
## amplitude_yaw_dumbbell            0     0.01274535     TRUE
TRUE
## kurtosis_yaw_forearm              0     0.00000000     TRUE
TRUE
## skewness_yaw_forearm              0     0.00000000     TRUE
TRUE
## amplitude_yaw_forearm             0     0.01274535     TRUE
TRUE
```

and which ones are the near-zero variance predictors:

```
x[x[,"zeroVar"] + x[,"nzv"] > 0, ]
##                           freqRatio percentUnique zeroVar
nzv
## new_window                 47.06531   0.016983696   FALSE
TRUE
## kurtosis_yaw_belt           0.00000   0.000000000    TRUE
TRUE
## skewness_yaw_belt           0.00000   0.000000000    TRUE
TRUE
## amplitude_yaw_belt          0.00000   0.008491848    TRUE
TRUE
## avg_roll_arm               54.00000   1.630434783   FALSE
TRUE
## stddev_roll_arm            54.00000   1.630434783   FALSE
TRUE
## var_roll_arm               54.00000   1.630434783   FALSE
TRUE
## avg_pitch_arm              54.00000   1.630434783   FALSE
TRUE
## stddev_pitch_arm           54.00000   1.630434783   FALSE
TRUE
## var_pitch_arm              54.00000   1.630434783   FALSE
TRUE
## avg_yaw_arm                54.00000   1.630434783   FALSE
TRUE
## stddev_yaw_arm             55.00000   1.621942935   FALSE
TRUE
```

```
## var_yaw_arm              55.00000   1.621942935   FALSE
TRUE
## amplitude_roll_arm       27.00000   1.587975543   FALSE
TRUE
## kurtosis_yaw_dumbbell     0.00000   0.000000000    TRUE
TRUE
## skewness_yaw_dumbbell     0.00000   0.000000000    TRUE
TRUE
## amplitude_yaw_dumbbell    0.00000   0.008491848    TRUE
TRUE
## kurtosis_yaw_forearm      0.00000   0.000000000    TRUE
TRUE
## skewness_yaw_forearm      0.00000   0.000000000    TRUE
TRUE
## amplitude_yaw_forearm     0.00000   0.008491848    TRUE
TRUE
## stddev_roll_forearm      52.00000   1.647418478   FALSE
TRUE
## var_roll_forearm         52.00000   1.647418478   FALSE
TRUE
## avg_pitch_forearm        49.00000   1.672894022   FALSE
TRUE
## stddev_pitch_forearm     24.50000   1.664402174   FALSE
TRUE
## var_pitch_forearm        49.00000   1.672894022   FALSE
TRUE
## avg_yaw_forearm          49.00000   1.672894022   FALSE
TRUE
## stddev_yaw_forearm       49.00000   1.672894022   FALSE
TRUE
## var_yaw_forearm          49.00000   1.672894022   FALSE
TRUE
y[y[,"zeroVar"] + y[,"nzv"] > 0, ]
##                         freqRatio percentUnique zeroVar
nzv
## new_window               47.73292    0.02549070   FALSE
TRUE
## kurtosis_yaw_belt         0.00000    0.00000000    TRUE
TRUE
## skewness_yaw_belt         0.00000    0.00000000    TRUE
TRUE
## amplitude_yaw_belt        0.00000    0.01274535    TRUE
TRUE
## avg_roll_arm             23.00000    1.77160336   FALSE
TRUE
```

```
## stddev_roll_arm         23.00000    1.77160336   FALSE
TRUE
## var_roll_arm            23.00000    1.77160336   FALSE
TRUE
## avg_pitch_arm           23.00000    1.77160336   FALSE
TRUE
## stddev_pitch_arm        23.00000    1.77160336   FALSE
TRUE
## var_pitch_arm           23.00000    1.77160336   FALSE
TRUE
## avg_yaw_arm             23.00000    1.77160336   FALSE
TRUE
## stddev_yaw_arm          25.00000    1.74611267   FALSE
TRUE
## var_yaw_arm             25.00000    1.74611267   FALSE
TRUE
## kurtosis_yaw_dumbbell    0.00000    0.00000000    TRUE
TRUE
## skewness_yaw_dumbbell    0.00000    0.00000000    TRUE
TRUE
## amplitude_yaw_dumbbell   0.00000    0.01274535    TRUE
TRUE
## kurtosis_yaw_forearm     0.00000    0.00000000    TRUE
TRUE
## skewness_yaw_forearm     0.00000    0.00000000    TRUE
TRUE
## amplitude_yaw_forearm    0.00000    0.01274535    TRUE
TRUE
## avg_roll_forearm        34.00000    1.63140454   FALSE
TRUE
## stddev_roll_forearm     35.00000    1.61865919   FALSE
TRUE
## var_roll_forearm        35.00000    1.61865919   FALSE
TRUE
## avg_pitch_forearm       34.00000    1.63140454   FALSE
TRUE
## stddev_pitch_forearm    34.00000    1.63140454   FALSE
TRUE
## var_pitch_forearm       34.00000    1.63140454   FALSE
TRUE
## avg_yaw_forearm         34.00000    1.63140454   FALSE
TRUE
## stddev_yaw_forearm      36.00000    1.60591384   FALSE
TRUE
```

```
## var_yaw_forearm          36.00000    1.60591384    FALSE
TRUE
```

1.- I will remove variables with nzv:

```
finalnzvtrain <-finaltrain[, -nearZeroVar(finaltrain)]
dim(finalnzvtrain)
## [1] 11776   132
finalnzvtest <-finaltest[, -nearZeroVar(finaltest)]
dim(finalnzvtest)
## [1] 7846  132
```

2.- In both data sets (finalnzvtrain and finalnzvtest) there are a lot of NA´s. I will remove variables that are mostly NA´s:

```
trainNA <- sapply(finalnzvtrain, function(x) mean(is.na(x)))
> 0.95
NoNAtrain <-finalnzvtrain[, trainNA==FALSE]
dim(NoNAtrain)
## [1] 11776    59
testNA <- sapply(finalnzvtest, function(x) mean(is.na(x))) >
0.95
NoNAtest <-finalnzvtest[, testNA==FALSE]
dim(NoNAtest)
## [1] 7846   59
```

3.- Having a look at the NoNAtrain and NoNAtest names, I will remove the columns (1:5) which seems to be identification variables.

```
trainclean<-NoNAtrain[, -(1:5)]
testclean<-NoNAtest[, -(1:5)]
dim(trainclean)
## [1] 11776    54
dim(testclean)
## [1] 7846   54
```

After performing the cleaning data process, we got two data subsets of 54 variables each.

4.- Processing validationdata and testclean data sets:

```
clean1 <- colnames(trainclean)
clean2 <- colnames(trainclean[, -54])  # remove the classe
column
```

```
testclean2 <- testclean[clean1]        # allow only variables
in testclean that are
#also in trainclean
validation2 <- validationdata[clean2]   # allow only
variables in validationdata that are also in trainclean
dim(testclean2)
## [1] 7846   54
dim(validation2)
## [1] 20 53
```

5.- Coerce the data into the same type:

```
for (i in 1:length(validation2) ) {
    for(p in 1:length(trainclean)) {
        if( length( grep(names(trainclean[i]), names
(validation2)[p]) ) == 1)  {
            class(validation2[p]) <- class(trainclean[i])
        }
    }
}
```

6.- Getting the same class between validation2 and trainclean:

```
validation3 <- rbind(trainclean[2, -54] , validation2)
validationf <- validation3[-1,]
```

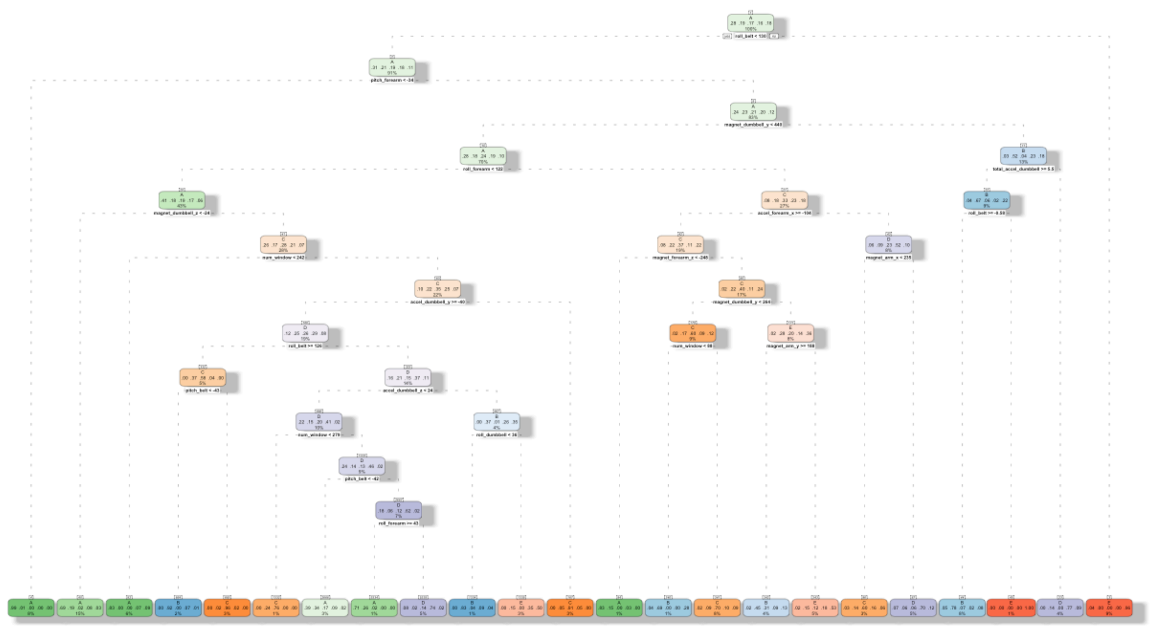
PREDICTION MODEL BUILDING

I will use three methods in the training data set (trainclean) to model the regressions and which one that is more accurate, I will apply to the testing set (validationf) and use it for the quiz prediction. These methods are: Decision Trees, Random Forests, and Generalized Boosted Model. Also, I will plot a Confusion Matrix to have a look at the accuracy of these models.

1.- PREDICTION WITH DECISION TREES

Fit the model:

```
set.seed(12345)
library(rpart)
library(rpart.plot)
library(rattle)
## Rattle: A free graphical interface for data mining with R.
## Versión 3.4.1 Copyright (c) 2006-2014 Togaware Pty Ltd.
## Escriba 'rattle()' para agitar, sacudir y  rotar sus
datos.
modDC <- rpart(classe ~ ., data=trainclean, method="class")
fancyRpartPlot(modDC)
```



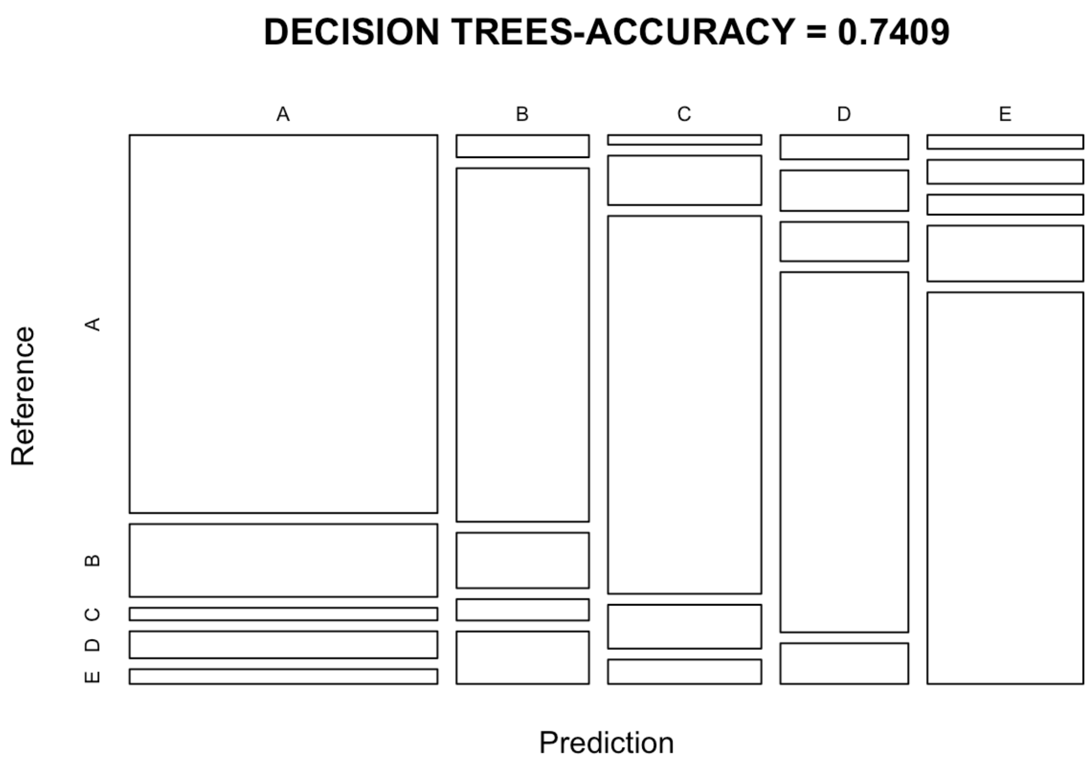
Rattle 2016-sep-22 14:25:14 apple

Prediction on Test data set (testclean2):

```
predictionDC <- predict(modDC, testclean2, type = "class")
confusionMatrix(predictionDC, testclean2$classe)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2061  397   68  147   80
##          B   52  829  130   50  123
##          C   26  134 1026  119   66
##          D   55   92   89  816   92
##          E   38   66   55  154 1081
##
## Overall Statistics
##
##                Accuracy : 0.7409
##                  95% CI : (0.731, 0.7506)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6695
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D
Class: E
## Sensitivity            0.9234   0.5461   0.7500   0.6345
0.7497
## Specificity            0.8767   0.9439   0.9467   0.9500
0.9511
## Pos Pred Value         0.7486   0.7002   0.7484   0.7133
0.7755
## Neg Pred Value         0.9664   0.8966   0.9472   0.9299
0.9440
## Prevalence             0.2845   0.1935   0.1744   0.1639
0.1838
## Detection Rate         0.2627   0.1057   0.1308   0.1040
0.1378
## Detection Prevalence   0.3509   0.1509   0.1747   0.1458
0.1777
## Balanced Accuracy      0.9001   0.7450   0.8484   0.7923
0.8504
conMatrixDC<-confusionMatrix(predictionDC, testclean2$classe)
```

PLOT MATRIX RESULTS:

```
plot(conMatrixDC$table, col = conMatrixDC$byClass,
     main = paste("DECISION TREES-ACCURACY =",
                  round(conMatrixDC$overall['Accuracy'], 4)))
```



2.-PREDICTION USING RANDOM FORESTS

Fit the model:

```
set.seed(12345)
ctrRF <- trainControl(method="cv", number=3,
verboseIter=FALSE)
modRF <- train(classe ~ ., data=trainclean, method="rf",
trControl=ctrRF)
## Loading required package: randomForest
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
modRF$finalModel
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 27
##
```

```
##         OOB estimate of  error rate: 0.25%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3348    0    0    0    0 0.000000000
## B    6 2266    5    1    1 0.005704256
## C    0    3 2047    4    0 0.003407984
## D    0    0    6 1924    0 0.003108808
## E    0    0    0    3 2162 0.001385681
```

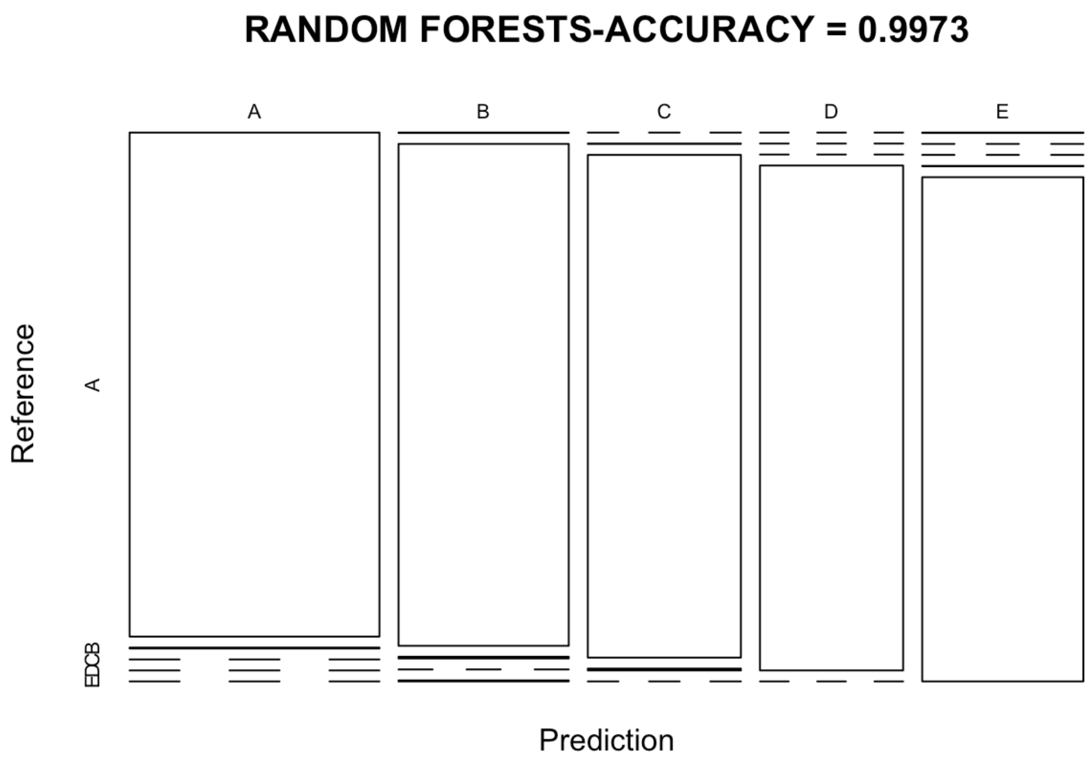
Prediction on Test data set (testclean2):

```
predictionRF <- predict(modRF, newdata=testclean2)
confusionMatrix(predictionRF, testclean2$classe)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2230    4    0    0    0
##          B    1 1513    5    0    3
##          C    0    1 1363    5    0
##          D    0    0    0 1280    0
##          E    1    0    0    1 1439
##
## Overall Statistics
##
##                Accuracy : 0.9973
##                  95% CI : (0.9959, 0.9983)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9966
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D
Class: E
## Sensitivity            0.9991   0.9967   0.9963   0.9953
0.9979
## Specificity            0.9993   0.9986   0.9991   1.0000
0.9997
## Pos Pred Value         0.9982   0.9941   0.9956   1.0000
0.9986
## Neg Pred Value         0.9996   0.9992   0.9992   0.9991
0.9995
```

```
## Prevalence             0.2845   0.1935   0.1744   0.1639
0.1838
## Detection Rate         0.2842   0.1928   0.1737   0.1631
0.1834
## Detection Prevalence   0.2847   0.1940   0.1745   0.1631
0.1837
## Balanced Accuracy      0.9992   0.9976   0.9977   0.9977
0.9988
conMatrixRF <-confusionMatrix(predictionRF,
testclean2$classe)
```

PLOT MATRIX RESULTS:

```
plot(conMatrixRF$table, col = conMatrixRF$byClass,
     main = paste("RANDOM FORESTS-ACCURACY =",
                  round(conMatrixRF$overall['Accuracy'], 4)))
```



3.- PREDICTION USING GENERALIZED BOOSTED MODEL

Fit the model:

```
set.seed(12345)
```

```
ctrGBM<- trainControl(method = "repeatedcv", number = 5,
repeats = 1)
modGBM  <- train(classe ~ ., data=trainclean, method = "gbm",
                    trControl = ctrGBM, 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
modGBM$finalModel
## A gradient boosted model with multinomial loss function.
## 150 iterations were performed.
## There were 53 predictors of which 40 had non-zero
influence.
```

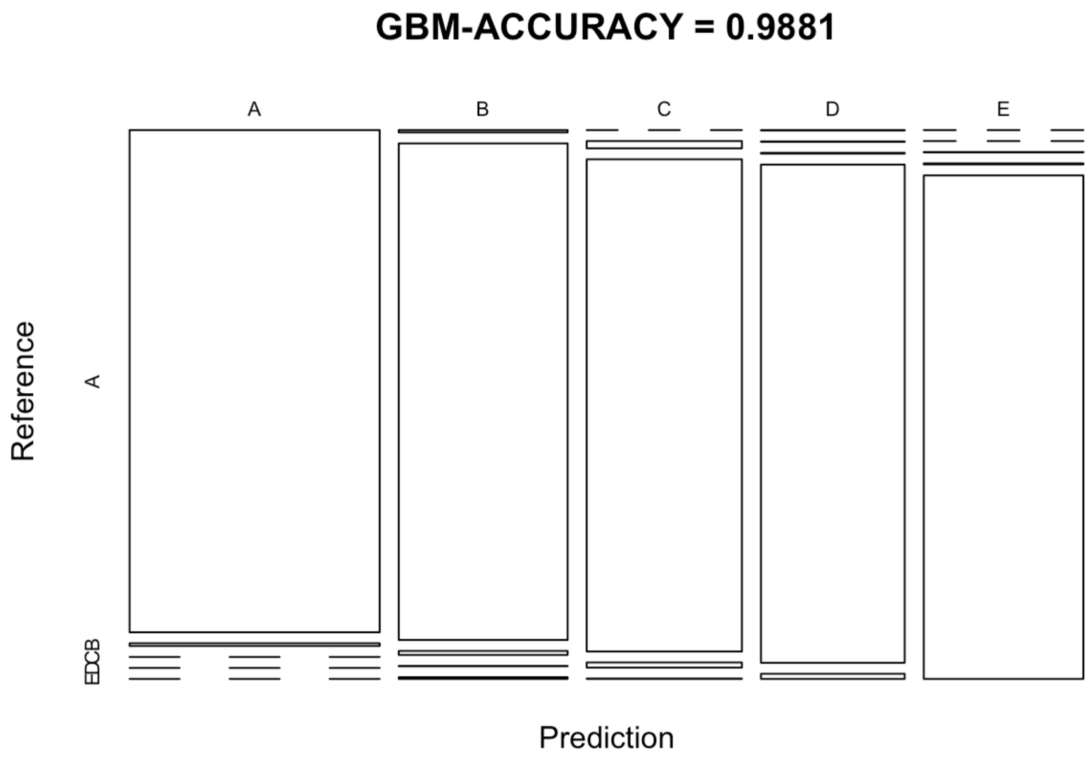
Prediction on Test data set (testclean2):

```
predictionGBM <- predict(modGBM, newdata=testclean2)
confusionMatrix(predictionGBM, testclean2$classe)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2224   12    0    0    0
##          B    7 1485   12    1    5
##          C    0   20 1353   14    1
##          D    1    1    2 1268   13
##          E    0    0    1    3 1423
##
## Overall Statistics
##
##                Accuracy : 0.9881
##                  95% CI : (0.9855, 0.9904)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.985
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D
Class: E
## Sensitivity            0.9964   0.9783   0.9890   0.9860
0.9868
## Specificity            0.9979   0.9960   0.9946   0.9974
0.9994
## Pos Pred Value         0.9946   0.9834   0.9748   0.9868
0.9972
## Neg Pred Value         0.9986   0.9948   0.9977   0.9973
0.9970
## Prevalence             0.2845   0.1935   0.1744   0.1639
0.1838
## Detection Rate         0.2835   0.1893   0.1724   0.1616
0.1814
## Detection Prevalence   0.2850   0.1925   0.1769   0.1638
0.1819
## Balanced Accuracy      0.9971   0.9872   0.9918   0.9917
0.9931
conMatrixGBM <-confusionMatrix(predictionGBM,
testclean2$classe)
```

PLOT MATRIX RESULTS:

```
plot(conMatrixGBM$table, col = conMatrixGBM$byClass,
     main = paste("GBM-ACCURACY =", round(conMatrixGBM
$overall['Accuracy'], 4)))
```



APPLYING THE SELECTED MODEL TO THE VALIDATION DATA

```
AccuracyModels<-data.frame(Model=c("DC", "RF", "GBM"),
Accuracy = rbind(conMatrixDC$overall[1], conMatrixRF$overall
[1], conMatrixGBM$overall[1]))
print(AccuracyModels)
##   Model  Accuracy
## 1    DC 0.7408871
## 2    RF 0.9973235
## 3   GBM 0.9881468
```

We can observe that Random Forest has a high accuracy (over 99.70%) and this is the higher of all of these models; cross validation is done with K=3 and the expected out -of-sample error is less than 0.3%. Therefore, I will apply the Random Forest method to the validation data set (validationf) to predict the 20 test cases:

Results (validation dataset):

```
predictionVAL <- predict(modRF, newdata=validationf)
predictionVAL
##  [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
```

Write the results to a text file for submission:

```
pml_write_files = function(x) {
        n = length(x)
        for(i in 1:n){
                filename = paste0("problem_id_",i,".txt")
                write.table(x[i], file = filename, quote =
FALSE, row.names = FALSE, col.names = FALSE)
        }
}

pml_write_files(predictionVAL)
```

CONCLUSION

Using exploratory analysis and combining different statistical models, our analysis suggests that our prediction function, developed using the Random Forests method with cross-validation, is be able to have a high accuracy (over 99.70%) to predict the 20 test cases (the manner in which the participants did the exercise) with 100% accuracy (20 points were awarded after submitting the 20 .txt files on the Course Project Submission). Random Forests is the more accurate method for our analysis after comparing it with other different methods such as Decision Trees and Generalized Boosted Model.