Creating a Prediction Model for Exercise Style Using Accelerometric Data

Thursday, February 19, 2015

Introduction

The following analysis is based on data from the 2013 Human Activity Recognition study by Wallace, Velloso and Fuks, **"Qualitative Activity Recognition of Weight Lifting Exercises"**. In it, the researchers record accelometric data from six subjects performing repetitions of the Unilateral Dumbbell Biceps Curl exercise. Each set of repetitions was done in five different ways. These are (as outlined in their paper):

- 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)
- Throwing the hips to the front (Class E)

Read more at: http://groupware.les.inf.puc-rio.br/har#ixzz3SNZVfYmr

The present paper utlises this data to identify a suitable model to be used to predict the manner in which the exercise was carried out (classes A-E), which is stored in the variable 'classe'.

Data Preparation

First install the required libraries that will be referenced during the analysis.

```r
suppressWarnings(library(caret))
```

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

```r
suppressWarnings(library(e1071))
suppressWarnings(library(corrplot))
suppressWarnings(library(survival))
```

```
## Loading required package: splines
##
## Attaching package: 'survival'
##
## The following object is masked from 'package:caret':
##
##     cluster
```

Next retrieve the data sets - note that the non-readable strings found in the data set are set to Null by the read.csv() function.

```r
setwd("C:/Training/DataScienceJHU/PracticalMachineLearning/Project")
testing <- read.csv("pml-testing.csv", stringsAsFactors=TRUE, na.strings=c("NA","","#DIV/0!"))
training <- read.csv("pml-training.csv", stringsAsFactors=TRUE, na.strings=c("NA","","#DIV/0!"))
```

Display the numbers of samples and variables for both the training and test data sets

```r
dim(training)
```

```
## [1] 19622   160
```

```r
dim(testing)
```

```
## [1]  20 160
```

Remove the null-only columns from the data set

```r
non_null_cols <- names(training)[apply(X=training, MARGIN=2, FUN=function(x) !sum(is.na(x[1])))]
trainingSubset <- training[,non_null_cols]
dim(trainingSubset)
```

```
## [1] 19622    60
```

100 variables have now been discarded, leaving the following 60 contenders.

```r
names(trainingSubset)
```

```
##  [1] "X"                    "user_name"            "raw_timestamp_part_1"
##  [4] "raw_timestamp_part_2" "cvtd_timestamp"       "new_window"
##  [7] "num_window"           "roll_belt"            "pitch_belt"
## [10] "yaw_belt"             "total_accel_belt"     "gyros_belt_x"
## [13] "gyros_belt_y"         "gyros_belt_z"         "accel_belt_x"
## [16] "accel_belt_y"         "accel_belt_z"         "magnet_belt_x"
## [19] "magnet_belt_y"        "magnet_belt_z"        "roll_arm"
## [22] "pitch_arm"            "yaw_arm"              "total_accel_arm"
## [25] "gyros_arm_x"          "gyros_arm_y"          "gyros_arm_z"
## [28] "accel_arm_x"          "accel_arm_y"          "accel_arm_z"
## [31] "magnet_arm_x"         "magnet_arm_y"         "magnet_arm_z"
## [34] "roll_dumbbell"        "pitch_dumbbell"       "yaw_dumbbell"
## [37] "total_accel_dumbbell" "gyros_dumbbell_x"     "gyros_dumbbell_y"
## [40] "gyros_dumbbell_z"     "accel_dumbbell_x"     "accel_dumbbell_y"
## [43] "accel_dumbbell_z"     "magnet_dumbbell_x"    "magnet_dumbbell_y"
## [46] "magnet_dumbbell_z"    "roll_forearm"         "pitch_forearm"
## [49] "yaw_forearm"          "total_accel_forearm"  "gyros_forearm_x"
## [52] "gyros_forearm_y"      "gyros_forearm_z"      "accel_forearm_x"
## [55] "accel_forearm_y"      "accel_forearm_z"      "magnet_forearm_x"
## [58] "magnet_forearm_y"     "magnet_forearm_z"     "classe"
```

We notice that some of the remaining variables are non-measurement data columns. These are now removed.

```r
measure_cols <- names(trainingSubset)[!(names(trainingSubset) %in%
                                          c("X",
                                            "user_name",
                                            "raw_timestamp_part_1",
                                            "raw_timestamp_part_2",
                                            "cvtd_timestamp",
                                            "new_window",
                                            "num_window"))]

trainingSubset <- trainingSubset[,names(trainingSubset) %in% measure_cols]
```

Determine if there are near zero covariates amonst the remaining variables.

```r
nzv <- nearZeroVar(trainingSubset, saveMetrics=TRUE)
nzv
```

```
##                      freqRatio percentUnique zeroVar   nzv
## roll_belt                1.102       6.77811   FALSE FALSE
## pitch_belt               1.036       9.37723   FALSE FALSE
## yaw_belt                 1.058       9.97350   FALSE FALSE
## total_accel_belt         1.063       0.14779   FALSE FALSE
## gyros_belt_x             1.059       0.71348   FALSE FALSE
## gyros_belt_y             1.144       0.35165   FALSE FALSE
## gyros_belt_z             1.066       0.86128   FALSE FALSE
## accel_belt_x             1.055       0.83580   FALSE FALSE
## accel_belt_y             1.114       0.72877   FALSE FALSE
## accel_belt_z             1.079       1.52380   FALSE FALSE
## magnet_belt_x            1.090       1.66650   FALSE FALSE
## magnet_belt_y            1.100       1.51870   FALSE FALSE
## magnet_belt_z            1.006       2.32902   FALSE FALSE
## roll_arm                52.338      13.52563   FALSE FALSE
## pitch_arm               87.256      15.73234   FALSE FALSE
## yaw_arm                 33.029      14.65702   FALSE FALSE
## total_accel_arm          1.025       0.33636   FALSE FALSE
## gyros_arm_x              1.016       3.27693   FALSE FALSE
## gyros_arm_y              1.454       1.91622   FALSE FALSE
## gyros_arm_z              1.111       1.26389   FALSE FALSE
## accel_arm_x              1.017       3.95984   FALSE FALSE
## accel_arm_y              1.140       2.73672   FALSE FALSE
## accel_arm_z              1.128       4.03629   FALSE FALSE
## magnet_arm_x             1.000       6.82397   FALSE FALSE
## magnet_arm_y             1.057       4.44399   FALSE FALSE
## magnet_arm_z             1.036       6.44685   FALSE FALSE
## roll_dumbbell            1.022      83.78351   FALSE FALSE
## pitch_dumbbell           2.277      81.22516   FALSE FALSE
## yaw_dumbbell             1.132      83.14137   FALSE FALSE
## total_accel_dumbbell     1.073       0.21914   FALSE FALSE
## gyros_dumbbell_x         1.003       1.22821   FALSE FALSE
## gyros_dumbbell_y         1.265       1.41678   FALSE FALSE
## gyros_dumbbell_z         1.060       1.04984   FALSE FALSE
## accel_dumbbell_x         1.018       2.16594   FALSE FALSE
## accel_dumbbell_y         1.053       2.37489   FALSE FALSE
## accel_dumbbell_z         1.133       2.08949   FALSE FALSE
```

```
## magnet_dumbbell_x        1.098        5.74865   FALSE FALSE
## magnet_dumbbell_y        1.198        4.30129   FALSE FALSE
## magnet_dumbbell_z        1.021        3.44511   FALSE FALSE
## roll_forearm            11.589       11.08959   FALSE FALSE
## pitch_forearm           65.983       14.85577   FALSE FALSE
## yaw_forearm             15.323       10.14677   FALSE FALSE
## total_accel_forearm      1.129        0.35674   FALSE FALSE
## gyros_forearm_x          1.059        1.51870   FALSE FALSE
## gyros_forearm_y          1.037        3.77637   FALSE FALSE
## gyros_forearm_z          1.123        1.56457   FALSE FALSE
## accel_forearm_x          1.126        4.04648   FALSE FALSE
## accel_forearm_y          1.059        5.11161   FALSE FALSE
## accel_forearm_z          1.006        2.95587   FALSE FALSE
## magnet_forearm_x         1.012        7.76679   FALSE FALSE
## magnet_forearm_y         1.247        9.54031   FALSE FALSE
## magnet_forearm_z         1.000        8.57711   FALSE FALSE
## classe                   1.470        0.02548   FALSE FALSE
```

It appears that there are none to be removed.

Determine which variables remain

```r
names(trainingSubset)
```

```
##  [1] "roll_belt"            "pitch_belt"           "yaw_belt"
##  [4] "total_accel_belt"     "gyros_belt_x"         "gyros_belt_y"
##  [7] "gyros_belt_z"         "accel_belt_x"         "accel_belt_y"
## [10] "accel_belt_z"         "magnet_belt_x"        "magnet_belt_y"
## [13] "magnet_belt_z"        "roll_arm"             "pitch_arm"
## [16] "yaw_arm"              "total_accel_arm"      "gyros_arm_x"
## [19] "gyros_arm_y"          "gyros_arm_z"          "accel_arm_x"
## [22] "accel_arm_y"          "accel_arm_z"          "magnet_arm_x"
## [25] "magnet_arm_y"         "magnet_arm_z"         "roll_dumbbell"
## [28] "pitch_dumbbell"       "yaw_dumbbell"         "total_accel_dumbbell"
## [31] "gyros_dumbbell_x"     "gyros_dumbbell_y"     "gyros_dumbbell_z"
## [34] "accel_dumbbell_x"     "accel_dumbbell_y"     "accel_dumbbell_z"
## [37] "magnet_dumbbell_x"    "magnet_dumbbell_y"    "magnet_dumbbell_z"
## [40] "roll_forearm"         "pitch_forearm"        "yaw_forearm"
## [43] "total_accel_forearm"  "gyros_forearm_x"      "gyros_forearm_y"
## [46] "gyros_forearm_z"      "accel_forearm_x"      "accel_forearm_y"
## [49] "accel_forearm_z"      "magnet_forearm_x"     "magnet_forearm_y"
## [52] "magnet_forearm_z"     "classe"
```

Look at the distribution of values in the classe variable that is to be predicted.

```r
table(trainingSubset$classe)
```

```
##
##    A    B    C    D    E
## 5580 3797 3422 3216 3607
```

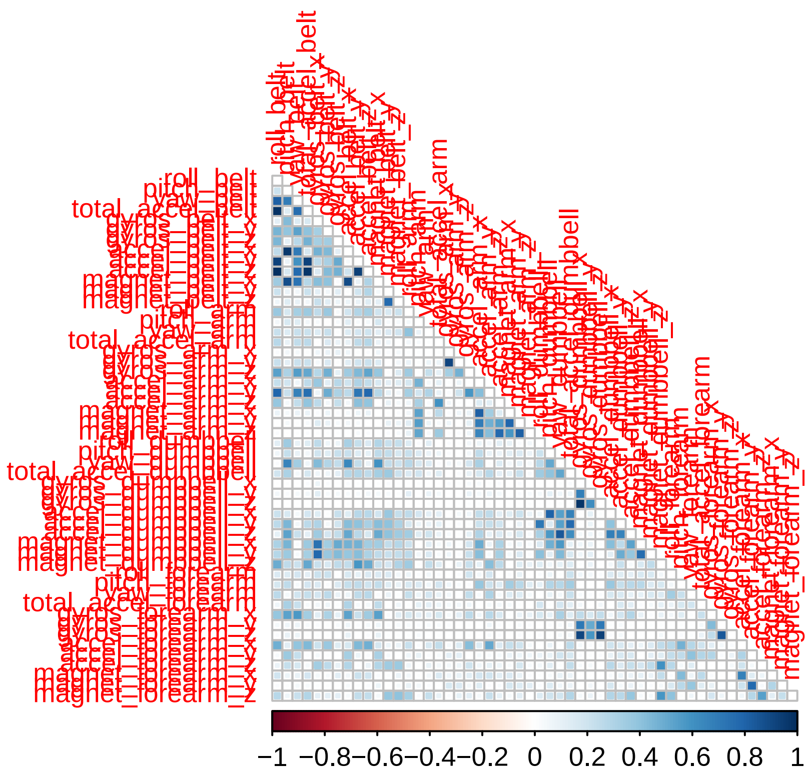
It appears that there are more in class A than any other class.

Determine any near-perfectly correlated variables which we define as those with a greater correlation than +/-0.8

```
correlValues <- abs(cor(trainingSubset[,-53]))
diag(correlValues) <- 0
```

Plot the correlations

```
corrplot(correlValues, method="square", type = "lower", tl.cex=0.8)
```



Determine the column indices of the variables with high correlation and display them

```
highCorr <- findCorrelation(correlValues[,-53], cutoff = .80)      # high correlation
names(trainingSubset[,highCorr])
```

```
##  [1] "accel_belt_z"     "roll_belt"        "accel_belt_y"
##  [4] "accel_dumbbell_z" "accel_belt_x"     "pitch_belt"
##  [7] "accel_arm_x"      "accel_dumbbell_x" "magnet_arm_y"
## [10] "gyros_arm_y"      "gyros_forearm_z"  "gyros_dumbbell_x"
```

Remove these highly correlated variables and display the remaining variables

```
trainingSubset<-trainingSubset[,-highCorr]
length(names(trainingSubset))
```

```
## [1] 41
```

Now divide and train the training data.

```
set.seed(56789)
inTrain = createDataPartition(y=trainingSubset$classe, p=0.75, list=FALSE)
traindata = trainingSubset[ inTrain,]
testdata =  trainingSubset[-inTrain,]
```

Setup the prediction model

We will evaluate three different models before settling on a final choice.

First we have elected to use a Linear Discriminant Analysis (LDA) model with cross-validation involving 3 times resampling

```
trainCtrl <- trainControl(method = "cv", number=3)
LDAmodelFit <- train(classe ~ ., method="lda", data=traindata, trControl=trainCtrl)
```

```
## Loading required package: MASS
```

```
LDAmodelFit
```

```
## Linear Discriminant Analysis
##
## 14718 samples
##    40 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
##
## Summary of sample sizes: 9812, 9811, 9813
##
## Resampling results
##
##   Accuracy  Kappa   Accuracy SD  Kappa SD
##   0.6417    0.5472  0.00488      0.006527
##
##
```

To see how successful the LDA model is we run a confusion matrix.

```
LDAConfusionmatrix <- confusionMatrix(testdata$classe,predict(LDAmodelFit,testdata))
LDAConfusionmatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1041  126  105  105   18
##          B  175  537  101   60   76
##          C   87   79  551  113   25
```

```
##          D   52   63  108  537   44
##          E   55  172  102   93  479
##
## Overall Statistics
##
##                Accuracy : 0.641
##                  95% CI : (0.628, 0.655)
##     No Information Rate : 0.288
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.546
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.738    0.550    0.570    0.591   0.7461
## Specificity             0.899    0.895    0.923    0.933   0.9010
## Pos Pred Value          0.746    0.566    0.644    0.668   0.5316
## Neg Pred Value          0.895    0.889    0.897    0.910   0.9593
## Prevalence              0.288    0.199    0.197    0.185   0.1309
## Detection Rate          0.212    0.110    0.112    0.110   0.0977
## Detection Prevalence    0.284    0.194    0.174    0.164   0.1837
## Balanced Accuracy       0.818    0.722    0.746    0.762   0.8235
```

The Accuracy of the model appears to be 64.1%.

The Out of Sample error is (1 - accuracy) = 35.9%

Next we have elected to use a Generalised Boosted Regression (GBM) model with cross-validation involving 3 times resampling

```r
trainCtrl <- trainControl(method = "cv", number=3)
GBMmodelFit <- train(classe ~ ., method="gbm", data=traindata, trControl=trainCtrl, verbose = FALSE)
```

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

```
## Warning: package 'gbm' was built under R version 3.1.2
```

```
## Loading required package: parallel
## Loaded gbm 2.1
## Loading required package: plyr
```

```r
GBMmodelFit
```

```
## Stochastic Gradient Boosting
##
## 14718 samples
##    40 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
```

```
##
## Summary of sample sizes: 9812, 9812, 9812
##
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy  Kappa   Accuracy SD  Kappa SD
##   1                   50      0.7208    0.6459  0.016621     0.021250
##   1                  100      0.7849    0.7276  0.011402     0.014519
##   1                  150      0.8245    0.7779  0.011277     0.014340
##   2                   50      0.8311    0.7860  0.010570     0.013304
##   2                  100      0.8864    0.8562  0.005535     0.006955
##   2                  150      0.9152    0.8927  0.004072     0.005082
##   3                   50      0.8726    0.8387  0.005707     0.007223
##   3                  100      0.9276    0.9084  0.004947     0.006214
##   3                  150      0.9493    0.9359  0.001123     0.001396
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3 and shrinkage = 0.1.
```

To see how successful the GBM model is we run a confusion matrix.

```
GBMConfusionmatrix <- confusionMatrix(testdata$classe,predict(GBMmodelFit,testdata))
GBMConfusionmatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1355   20    3   17    0
##          B   34  865   42    4    4
##          C    0   43  797   15    0
##          D    1    3   34  753   13
##          E    3    7    3    6  882
##
## Overall Statistics
##
##                Accuracy : 0.949
##                  95% CI : (0.942, 0.955)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.935
##  Mcnemar's Test P-Value : 4.16e-05
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.973    0.922    0.907    0.947    0.981
## Specificity             0.989    0.979    0.986    0.988    0.995
## Pos Pred Value          0.971    0.911    0.932    0.937    0.979
## Neg Pred Value          0.989    0.982    0.980    0.990    0.996
## Prevalence              0.284    0.191    0.179    0.162    0.183
```

```
## Detection Rate          0.276    0.176    0.163    0.154    0.180
## Detection Prevalence    0.284    0.194    0.174    0.164    0.184
## Balanced Accuracy       0.981    0.950    0.946    0.967    0.988
```

The Accuracy of the model appears to be 94.9%.

The Out of Sample error is (1 - accuracy) = 5.1%

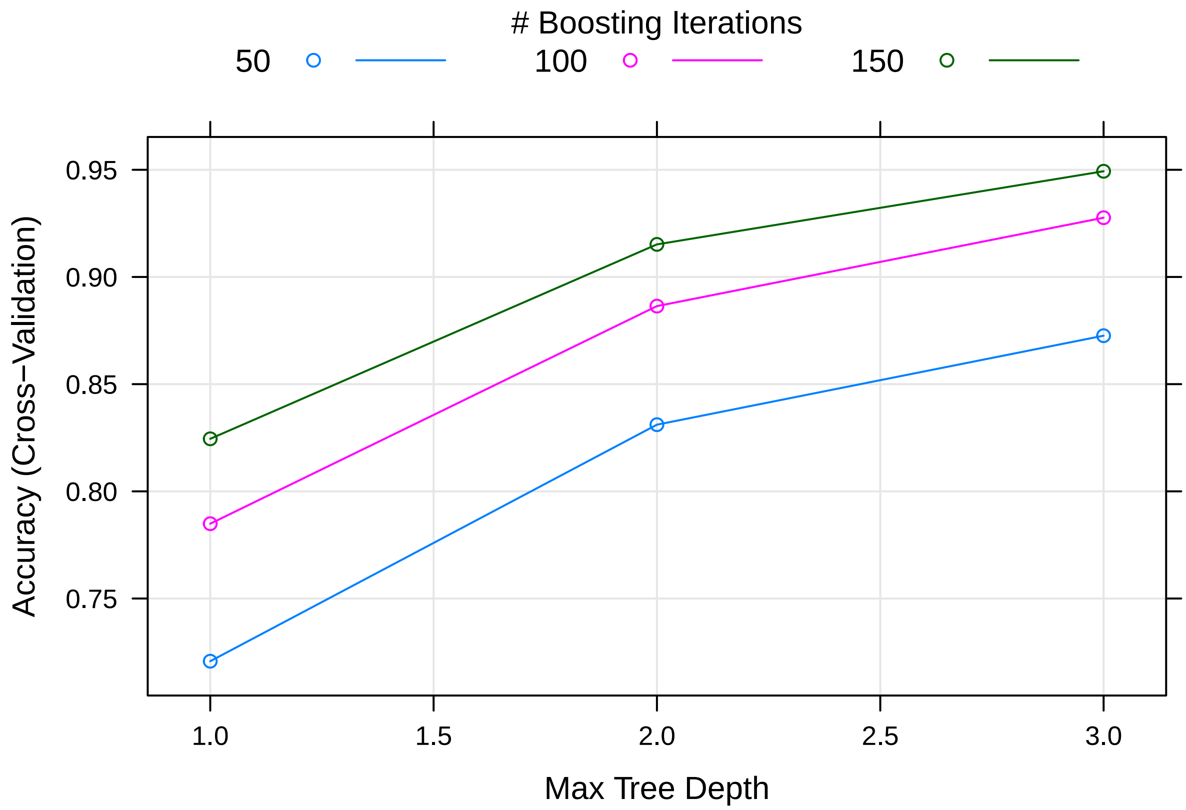
Lets see what the optimal model parameters were.

```r
GBMmodelFit
```

```
## Stochastic Gradient Boosting
##
## 14718 samples
##    40 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
##
## Summary of sample sizes: 9812, 9812, 9812
##
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy  Kappa   Accuracy SD  Kappa SD
##   1                   50      0.7208    0.6459  0.016621     0.021250
##   1                  100      0.7849    0.7276  0.011402     0.014519
##   1                  150      0.8245    0.7779  0.011277     0.014340
##   2                   50      0.8311    0.7860  0.010570     0.013304
##   2                  100      0.8864    0.8562  0.005535     0.006955
##   2                  150      0.9152    0.8927  0.004072     0.005082
##   3                   50      0.8726    0.8387  0.005707     0.007223
##   3                  100      0.9276    0.9084  0.004947     0.006214
##   3                  150      0.9493    0.9359  0.001123     0.001396
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3 and shrinkage = 0.1.
```

These results are reflected in the following plot.

```r
plot(GBMmodelFit)
```



It appears that greater accuracy is achieved in the gbm model with more trees and greater depth of the analysis. An even better result may be had by changing either of these parameters.

Finally we have elected to use a Random Forest (RF) model with cross-validation involving 3 times resampling

```
trainCtrl <- trainControl(method = "cv", number=3)
RFmodelFit <- train(classe ~ ., method="rf", data=traindata, trControl=trainCtrl, importance=TRUE)
```

```
## Loading required package: randomForest
```

```
## Warning: package 'randomForest' was built under R version 3.1.2
```

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
RFmodelFit
```

```
## Random Forest
##
## 14718 samples
##    40 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
```

```
##
## Summary of sample sizes: 9811, 9812, 9813
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa   Accuracy SD  Kappa SD
##    2    0.9887    0.9856  0.0023602    0.0029875
##   21    0.9882    0.9851  0.0011960    0.0015138
##   40    0.9829    0.9784  0.0004235    0.0005373
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

To see how successful the RF model is we run a confusion matrix.

```
RFConfusionmatrix <- confusionMatrix(testdata$classe,predict(RFmodelFit,testdata))
RFConfusionmatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1392    3    0    0    0
##          B    6  938    5    0    0
##          C    0    9  844    2    0
##          D    0    0   14  789    1
##          E    0    0    0    0  901
##
## Overall Statistics
##
##                Accuracy : 0.992
##                  95% CI : (0.989, 0.994)
##     No Information Rate : 0.285
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.99
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.996    0.987    0.978    0.997    0.999
## Specificity             0.999    0.997    0.997    0.996    1.000
## Pos Pred Value          0.998    0.988    0.987    0.981    1.000
## Neg Pred Value          0.998    0.997    0.995    1.000    1.000
## Prevalence              0.285    0.194    0.176    0.161    0.184
## Detection Rate          0.284    0.191    0.172    0.161    0.184
## Detection Prevalence    0.284    0.194    0.174    0.164    0.184
## Balanced Accuracy       0.997    0.992    0.988    0.997    0.999
```

Lets see what the estimated overall error rate is:

```r
RFmodelFit$finalModel
```

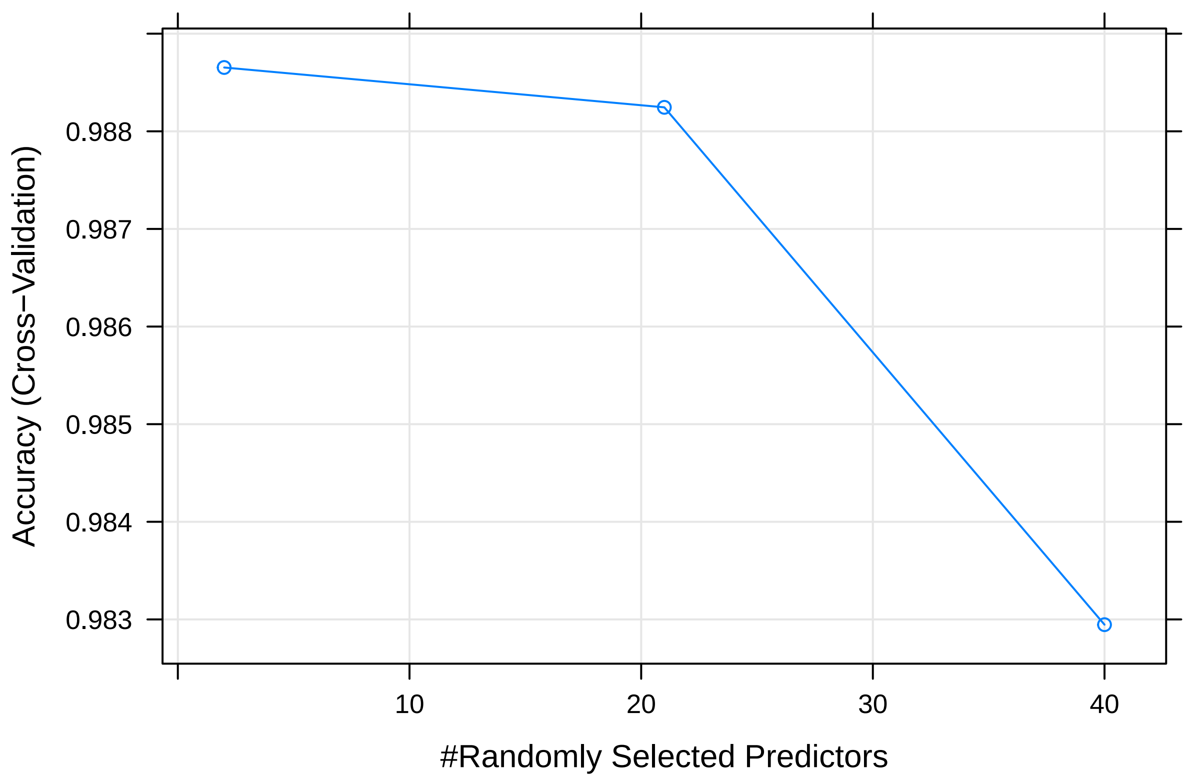
```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 0.74%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4181    3    0    0    1   0.0009558
## B   18 2822    8    0    0   0.0091292
## C    0   19 2544    4    0   0.0089599
## D    0    0   48 2362    2   0.0207297
## E    0    1    0    5 2700   0.0022173
```

The accuracy is: 99.2%.

The Out of Sample error is (1 - accuracy) = 0.8

This is reflected in the following plot.

```r
plot(RFmodelFit)
```



It appears that the RF model is extremely accurate with over 99% accuracy across each classe grouping. This was closely followed by the gbm boosting model at just under 95% accuracy.The LDA model followed way behind with only a 64% accuracy.

In Conclusion

So what are the answers given by each of the models?

Linear Discriminant Analysis:

```
LDAtest_answers <- predict(LDAmodelFit, testingSubset[,-41])
```

GBM:

```
GBMtest_answers <- predict(GBMmodelFit, testingSubset[,-41])
```

Random Forest:

```
RFtest_answers <- predict(RFmodelFit, testingSubset[,-41])
```

Let's look at the results of each test when the prediction models are applied:

```
answers <- as.data.frame(matrix(seq(NA), nrow=3, ncol=21))

names(answers)<-c('Test Name',
                  'Test 1','Test 2','Test 3','Test 4','Test 5',
                  'Test 6','Test 7','Test 8','Test 9','Test 10',
                  'Test 11','Test 12','Test 13','Test 14','Test 15',
                  'Test 16','Test 17','Test 18','Test 19','Test 20')

answers[1,1] <- 'LDA'
answers[2,1] <- 'GMB'
answers[3,1] <- 'RF'

answers[1,2:21] <- as.factor(LDAtest_answers)
answers[2,2:21] <- as.factor(GBMtest_answers)
answers[3,2:21] <- as.factor(RFtest_answers)

answers
```

```
##   Test Name Test 1 Test 2 Test 3 Test 4 Test 5 Test 6 Test 7 Test 8 Test 9
## 1       LDA      B      A      A      C      C      C      D      D      A
## 2       GMB      B      A      B      A      A      C      D      B      A
## 3        RF      B      A      B      A      A      E      D      B      A
##   Test 10 Test 11 Test 12 Test 13 Test 14 Test 15 Test 16 Test 17 Test 18
## 1       A       D       A       B       A       E       A       A       B
## 2       A       B       C       B       A       E       E       A       B
## 3       A       B       C       B       A       E       E       A       B
##   Test 19 Test 20
## 1       B       B
## 2       B       B
## 3       B       B
```

From these results it appears that the minor difference in accuracy between the RF and GMB models made no difference in the outcome when applied to the test set. The LDA model, however, produced vastly different results.

On the basis of accuracy alone, the present analysis has decided to choose the Random Forest model.