Predicting How Well An Exercise Is Done

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, the goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants and Predict how well an exercise is done. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways.

The data for this project and its description come from this source: http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har.

Data Analysis:

The flow of data analysis is broadly into 5 stages: loading, cleaning splicing, fitting and predicting. They are detailed as below:

1. Loading Data

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

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

The data is downloaded into the working directory. As the data is in .csv format, read.csv ( ) is used to load the training data and quiz testing data.

```
training <- read.csv ("pml-training.csv")
quiztesting <- read.csv ("pml-testing.csv")
```

To predict the manner in which they did the exercise, the “classe” variable in the training set should be predicted using any of the other variables.

2. Cleaning and Selection of Features

Basic exploratory analysis can be done using the functions head ( ), tail ( ), str ( ), summary ( ) etc. It is ignored here due to space constraints. It reveals that lot of columns have NAs. These variables do not add value to the model fit and are to be removed. Since complete.cases ( ) gives observations/rows which have no missing values across the entire sequence, we cannot use it for columns here. So is.na ( ) is used over the data sets and column sums is calculted for the presence of NAs. This condition is used to subset the original data.

```
training2 <- training [ ,colSums(is.na(training)) == 0]
quiztesting2 <- quiztesting [ ,colSums(is.na(quiztesting)) == 0]
```

Now viewing the data columns reveals that the first few columns are variables for understanding purpose

and that they are not the independent variables upon which classe depends. So they are removed from the above data sets.

```
training3 <- training2 [ ,8:93]
quiztesting3 <- quiztesting2 [ ,8:60]
```

There are still a lot of variables in the data (using the dim ( ) function we can find the dimensions of the dataframe). So we can find the less impacting variables using the nearZeroVar ( ) function in caret package and remove them from the above training data set by subsetting.

```
library (caret)
```

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

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

```
nzv <- nearZeroVar(training3)
training4 <- training3 [ ,-nzv]
```

```
dim (training4)
```

```
## [1] 19622    53
```

```
dim (quiztesting3)
```

```
## [1] 20 53
```

Now both of our original datasets come down to 52 independent variables upon which classe depends.

3. Data Splicing

Data Splicing can be done by using the function createDataPartition ( ) in the caret package. The training4 dataset is partitioned, by subsetting, into training5 and testing data sets for fitting and validating respectively.

```
partition <- createDataPartition(y=training4$classe, p = 0.8, list = FALSE)
training5 <- training4 [partition, ]
testing <- training4 [-partition, ]
```

4. Fitting a model

There are many methods for model fitting viz., linear method, general linear method, classification trees, bagging, random forests, boosting, model based method etc. The random forest method is chosen as it was suggested by the instructor as the most widey used model fit in the Data Scientists community for its versatility and accuracy.

A simple fit using the train ( ) function with just the variables info, input data and method type was tried. But it was taking a lot of time for execution.

To improve performance of random forest in caret, parallel implementation as suggested by Len Greski was used. In this process

- First the Cores in the computer are detected and one minus available is alloted for the process. The detectCores ( ) function, makeCluster ( ), registerDoParallel ( ) in the parallel and doParallel packages are used.
- Secondly, trainControl ( ) function is configured with Resampling method as K- Fold Cross Validation with 5 folds. (My laptop has similar configuration as the tested HP Envy X2 tablet. So to half the time of tested 74 minutes, I decreased number of folds from 10 to 5. It took around 35 minutes. However the accuracy was targeted for 99% and it was acheived.)
- Now the model is fit using the train ( ) in caret package
- Finally cores allocation is stopped using the stopCluster ( ) and registerDoSEQ ( ) functions.

A detailed and good explanation of the process is available here:
https://github.com/lgreski/datasciencectacontent/blob/master/markdown/pml-randomForestPerformance.md

```
set.seed (2608)
library(parallel)
library(doParallel)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
cluster <- makeCluster(detectCores() - 1) # convention to leave 1 core for OS
registerDoParallel(cluster)

fitControl <- trainControl(method = "cv", number = 5, allowParallel = TRUE)

fit <- train (classe ~ ., data = training5, method="rf", trControl = fitControl)
```

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

```
stopCluster(cluster)
registerDoSEQ()
```

The details and accuracy of the above model fit can be found by extracting the finalModel parameter on fit.

```
fit$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: 0.55%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 4460    3    0    0    1 0.0008960573
## B   20 3013    4    1    0 0.0082290981
## C    0    9 2718   11    0 0.0073046019
## D    0    1   23 2547    2 0.0101049359
## E    0    2    4    5 2875 0.0038115038
```

Thus the accuracy of the fit was > 99% and In Sample Error (OOB estimate of error rate) was < 1%

5. Predicting using the model

Using the above model fit, we can predict the classe values of above partitioned testing data to find the Out Sample Error. The predict ( ) fucntion and confusionMatrix ( ) functions in the caret package are used to predict and find accuracy respectively.

```
pred <- predict (fit, newdata = testing)
confusionMatrix(testing$classe, pred)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1116    0    0    0    0
##          B    4  754    1    0    0
##          C    0    1  681    2    0
##          D    0    0    7  635    1
##          E    0    0    0    0  721
##
## Overall Statistics
##
##                Accuracy : 0.9959
##                  95% CI : (0.9934, 0.9977)
##     No Information Rate : 0.2855
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9948
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9964   0.9987   0.9884   0.9969   0.9986
## Specificity            1.0000   0.9984   0.9991   0.9976   1.0000
## Pos Pred Value         1.0000   0.9934   0.9956   0.9876   1.0000
## Neg Pred Value         0.9986   0.9997   0.9975   0.9994   0.9997
## Prevalence             0.2855   0.1925   0.1756   0.1624   0.1840
## Detection Rate         0.2845   0.1922   0.1736   0.1619   0.1838
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9982   0.9985   0.9937   0.9972   0.9993
```

The accuracy of the prediction was > 99% and Out Sample Error was < 1%

Results

Now we can predict the quiz testing data using our model fit similar to the above step using the predict ( ) fucntion. The predicted values are printed below in the same sequence.

```
quizpred <- predict (fit, newdata = quiztesting3)
quizpred
```

```
##  [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
```

Appendix:

Figure 1: Plotting the fit

```
plot (fit)
```

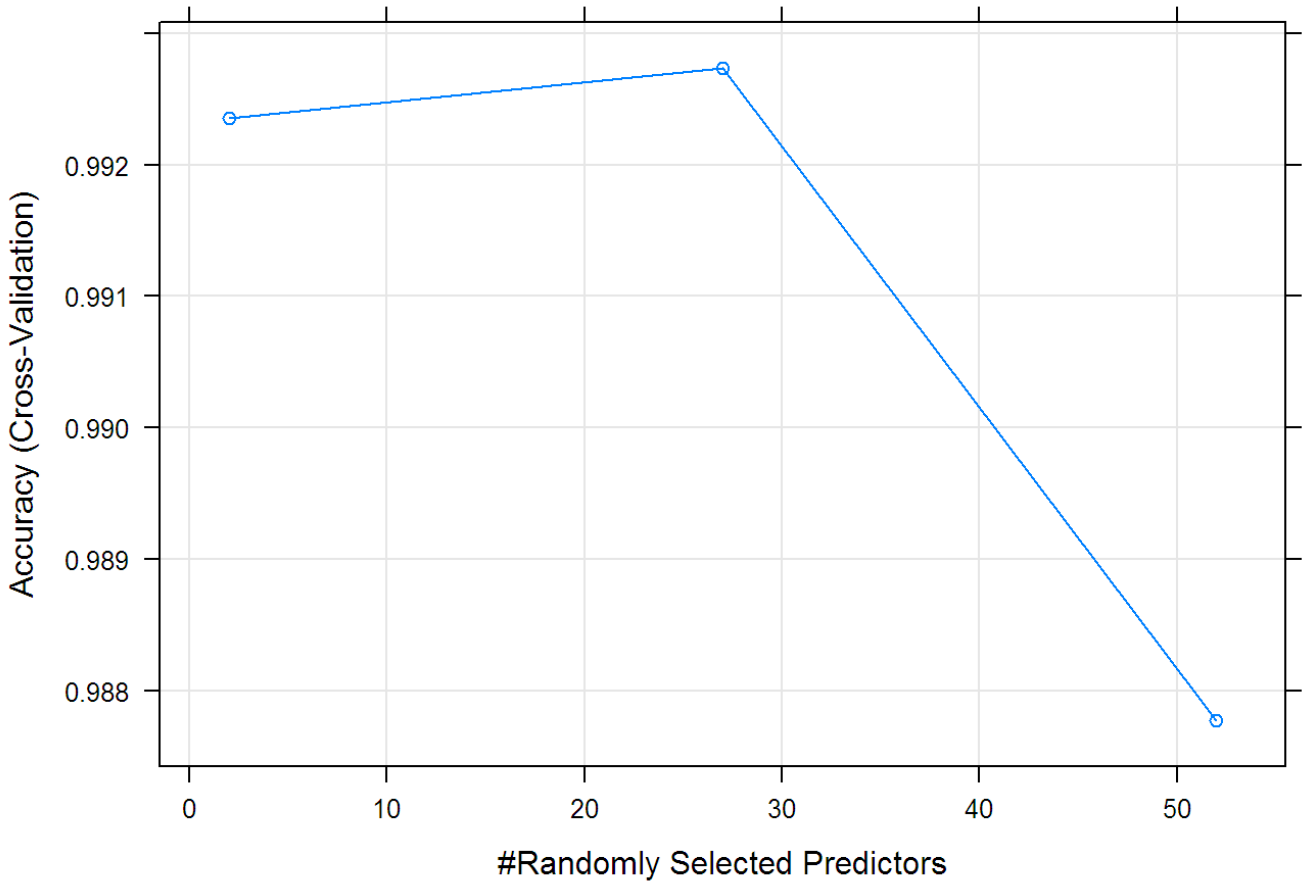
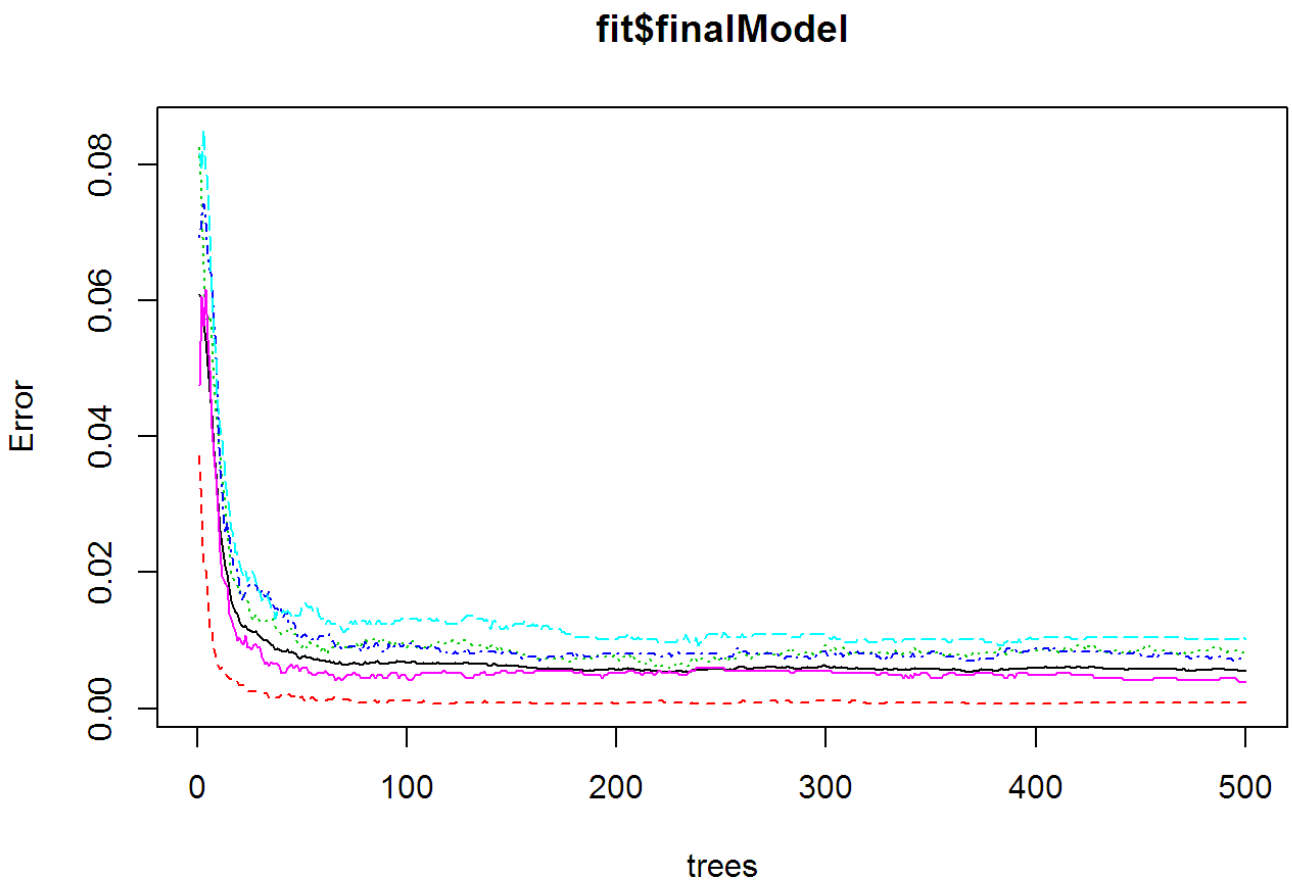


Figure 2: Plotting the finalModel

```
plot (fit$finalModel)
```



Getting a single tree:

To view a single tree we can use the R Code: getTree (fit$finalModel, k=1). It is ignored here due to space constraints.