Coursera Assignment for Practical Machine Learning

The goal of the project is to predict the manner in which people did an exercise using a large amount of data about personal activity measured using devices such as Jawbone Up, Fitbit, etc.

Data for the exercise was downloaded from links provided on the course assignment website. The variable to be predicted is “classe” in the training set using an approriate model and cross validation dataset. Furthermore, we are to use the model to predict 20 different test cases.

We first load the training and testing datasets. We also reatin only those columns which do not have any missing data. We also drop a few variables like timestamp etc. which we do not believe to be useful in prediction

```
setwd("C:\\Personal1\\Learning\\Coursera\\Data Science Specialization Track\\8. Practical Machine Learning\\Course Project\\Data")


TrainData <- read.csv("pml-training.csv", head = TRUE, na.strings = c("","#DIV/0!","NA"))

TestData <- read.csv("pml-testing.csv", head = TRUE, na.strings = c("","#DIV/0!","NA"))


TrainData <- TrainData[ , apply(TrainData, 2, function(x) !any(is.na(x)))]

TestData <- TestData[ , apply(TestData, 2, function(x) !any(is.na(x)))]


Drop <- c("X", "user_name", "raw_timestamp_part_1", "raw_timestamp_part_2", "cvtd_timestamp", "new_window")


TrainData <- TrainData[,!(names(TrainData) %in% Drop)]

TestData <- TestData[,!(names(TestData) %in% Drop)]
```

We now have about 50+ variables in the data and the correlation matrix shows a lot of very high correlations which indicates redudUncy in the variables. As such a few of these variables may do a good job in predicting the outcome. We can use a Principal Compoments preprocessing step to achieve this. Another approach is to actually conduct a Factor Analyis and examine how the variables load into factors. This gives us information about using a few proxy variables to include in the final analysis (The second approach is especially useful if the researcher knows about how the variables are related to each other. This helps identify groups of variables that jointly represent a theme)

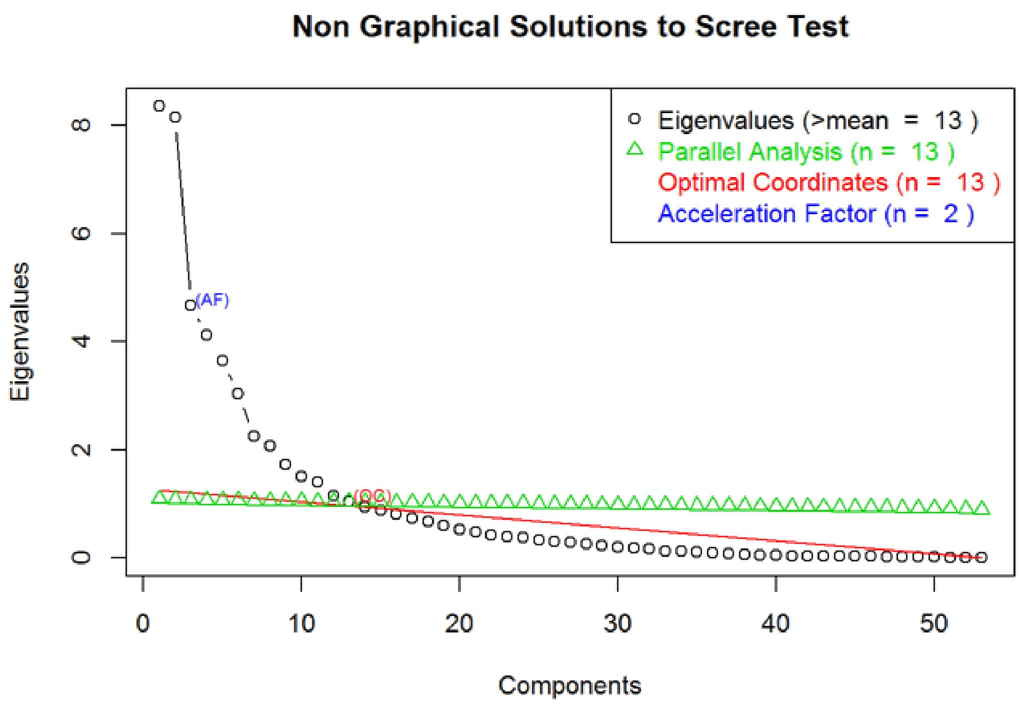
We use the eigen value cutoff method to determine that 12 Principal Components can be used to reduce the dataset. We then perform a Principal Components Analysis extracting 12 components and retain the variable with the highest loading in each component.

```
DataforPC <- TrainData[,1:53]


library(nFactors)
```

```
## Loading required package: MASS

## Loading required package: psych

## Loading required package: boot

##

## Attaching package: 'boot'

##

## The following object is masked from 'package:psych':

##

##     logit

##

## Loading required package: lattice

##

## Attaching package: 'lattice'

##

## The following object is masked from 'package:boot':

##

##     melanoma

##

##

## Attaching package: 'nFactors'

##

## The following object is masked from 'package:lattice':
```

```
##
##     parallel
```

```
ev <- eigen(cor(DataforPC))
ap <- parallel(subject = nrow(DataforPC),var = ncol(DataforPC), rep = 100, cent = .05)
nS <- nScree(x = ev$values, aparallel = ap$eigen$qevpea)
plotnScree(nS)
```



```
library(psych)
PCAnalysis <- principal(DataforPC, nfactors = 12, rotate = "varimax")
```

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

We retain the following 12 variables for our analaysis:

1. roll_belt
2. pitch_belt
3. magnet_arm_y
4. gyros_forearm_z
5. magnet_belt_z
6. magnet_forearm_y

7. gyros_arm_x
8. magnet_forearm_x
9. accel_dumbbell_x
10. accel_dumbbell_y
11. total_accel_arm
12. magnet_dumbbell_y

```
Keep <- c("roll_belt", "pitch_belt", "magnet_arm_y", "gyros_forearm_z", "magnet_belt_z
", "magnet_forearm_y", "gyros_arm_x", "magnet_forearm_x", "accel_dumbbell_x", "accel_d
umbbell_y",

"total_accel_arm", "magnet_dumbbell_y", "classe")



TrainData <- TrainData[,(names(TrainData) %in% Keep)]

TestData <- TestData[,(names(TestData) %in% Keep)]
```

Given the size of the dataset and processing time on the computer being used, we decided to use a smaller sample of the training data for the model and a smaller section of the remaining data for validation.

We first create these datasets. We choose 20% of the training dataset for training and 20% of the remaining (i.e. 20% of the 80% which was left after extracting the training dataset) for validation

```
library(AppliedPredictiveModeling)

library(caret)
```

```
## Loading required package: ggplot2

##

## Attaching package: 'ggplot2'

##

## The following object is masked from 'package:psych':

##

##     %+%
```

```
set.seed(975)



TrainIndex = createDataPartition(y = TrainData$classe, p = 0.2, list = FALSE)



TrainData = TrainData[TrainIndex,]
```

```
ValidateData = TrainData[-TrainIndex,]


set.seed(975)


ValidateIndex = createDataPartition(y = ValidateData$classe, p = 0.2, list = FALSE)

ValidateData = ValidateData[ValidateIndex,]
```

We also implement some code to enable parallel processing to address processing time and memory issues we faced on our computer. We then train the dataset using Random Forests.

```
library(doParallel)
```

```
## Loading required package: foreach

## Loading required package: iterators

## Loading required package: parallel
```

```
cl <- makeCluster(detectCores())

registerDoParallel(cl)


ModelFit<- train(classe ~ ., data = TrainData, method = "rf", prox = TRUE)
```

```
## Loading required package: randomForest

## randomForest 4.6-10

## Type rfNews() to see new features/changes/bug fixes.
```

```
ModelFit
```

```
## Random Forest

##

## 3927 samples

##   12 predictors

##    5 classes: 'A', 'B', 'C', 'D', 'E'

##

## No pre-processing

## Resampling: Bootstrapped (25 reps)

##
```

```
## Summary of sample sizes: 3927, 3927, 3927, 3927, 3927, 3927, ...
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
##   2     0.9       0.9    0.008        0.01
##   7     0.9       0.9    0.01         0.01
##   10    0.9       0.9    0.01         0.01
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```
confusionMatrix(ValidateData$classe, predict(ModelFit,ValidateData))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 181   0   0   0   0
##          B   0 122   0   0   0
##          C   0   0 109   0   0
##          D   0   0   0 105   0
##          E   0   0   0   0 116
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.994, 1)
##     No Information Rate : 0.286
##     P-Value [Acc > NIR] : <2e-16
##
```

```
##                   Kappa : 1

##  Mcnemar's Test P-Value : NA

##

## Statistics by Class:

##

##                      Class: A Class: B Class: C Class: D Class: E

## Sensitivity             1.000    1.000    1.000    1.000    1.000

## Specificity             1.000    1.000    1.000    1.000    1.000

## Pos Pred Value          1.000    1.000    1.000    1.000    1.000

## Neg Pred Value          1.000    1.000    1.000    1.000    1.000

## Prevalence              0.286    0.193    0.172    0.166    0.183

## Detection Rate          0.286    0.193    0.172    0.166    0.183

## Detection Prevalence    0.286    0.193    0.172    0.166    0.183

## Balanced Accuracy       1.000    1.000    1.000    1.000    1.000
```

The training data shows an accuracy of 92% which is pretty decent given that we are using only a small subset of the variables and also the data. We expect the out of sample error in the cross validation sample to be correspondingly good which we verify using the Confusion Matrix. In fact it is flawless with a 100% accuracy rate.

We finally predict the 20 test cases. Based on the feedback from the course assignment submission page, the accuracy is 95% (19 oit of 20 cases were correctly predicted)

```
Predict <- predict(ModelFit, TestData)

Predict
```

```
##  [1] B A B A A E D B A A A C B A E E A B B B

## Levels: A B C D E
```