Practical Machine Learning Project: Prediction Assignment Writeup

Chandani Patel

06-April-2020

Overview

This document is created for final project submission for Courseara course `Practical Machine Learning` which is part of Machine learning specialization course. There is data of human activity is provided by courseara which contains accelerometers data on the belt, forearm, arm, and dumbell of 6 participants like the acticity people regularly do, how much of a particular activity they do, how well they do it. The Project is to predict the manner in which they did exercise.

Tools & Languages

- R
- R Studio
- Caret package

Problem Statement

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, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. 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).

Datasets

- Training Data: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv
- Test Data: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

Load the libraries

```r
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```r
library(rpart)
library(rpart.plot)
library(RColorBrewer)
library(rattle)
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.3.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```r
library(corrplot)
```

```
## corrplot 0.84 loaded
```

Load Data from CSV & Cleaning the data

This step will load data from csv to dataframe. Following steps as part of this process * Load the data from the CSV * Clean data: remove the variables which has missing values, this will reduce the dataset

```r
# Load Data sets
trainingData <- read.csv("pml-training.csv", header = TRUE)
validationData <- read.csv("pml-testing.csv", header = TRUE)

# Clean Data
trainData<- trainingData[, colSums(is.na(trainingData)) == 0]
validData <- validationData[, colSums(is.na(validationData)) == 0]
dim(trainData)
```

```
## [1] 19622    93
```

```r
dim(validData)
```

```
## [1] 20 60
```

```r
# first seven variables have little impact on the outcome classe, let's remove
trainData <- trainData[, -c(1:7)]
validData <- validData[, -c(1:7)]

# Print Data
dim(trainData)
```

```
## [1] 19622    86
```

```r
dim(validData)
```

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

Splitting of Data

There will be 2 datasets,

- training data: this will be 70% of provided training data and used to train the model
- test data: this will be 30% of provided training data and used to test data and quiz results

```r
set.seed(1234)
inTrain <- createDataPartition(trainData$classe, p = 0.7, list = FALSE)
trainData <- trainData[inTrain, ]
```

```r
testData <- trainData[-inTrain, ]
dim(trainData)
```

```
## [1] 13737    86
```

Clean Data with near-zero-variance

```r
NZV <- nearZeroVar(trainData)
trainData <- trainData[, -NZV]
testData  <- testData[, -NZV]
dim(trainData)
```

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

```r
dim(testData)
```

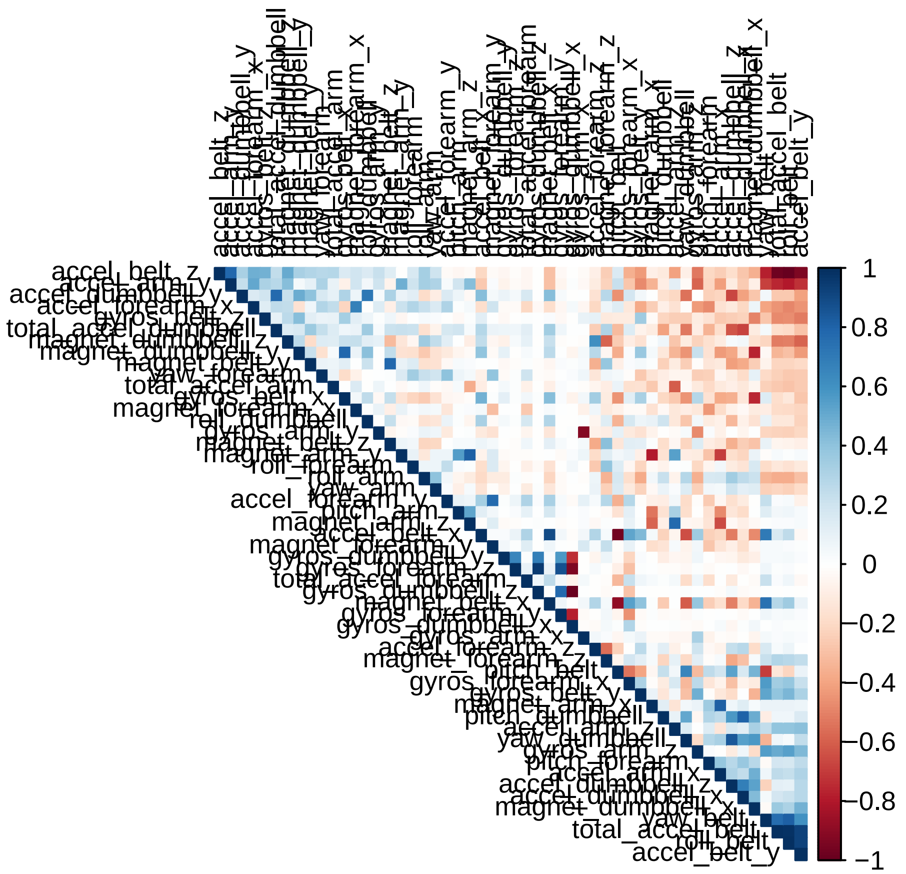
```
## [1] 4123   53
```

After above step the dataset will have 53 vaeiables.

Correlation plot

Let's put the corelation plot uses the following parameters.

```r
cor_mat <- cor(trainData[, -53])
corrplot(cor_mat, order = "FPC", method = "color", type = "upper",
         tl.cex = 0.8, tl.col = rgb(0, 0, 0))
```



In the above graph the correlated predictors (variables) are those with a dark color intersection.

Find the names of variables

```
highlyCorrelated = findCorrelation(cor_mat, cutoff=0.75)
names(trainData)[highlyCorrelated]
```

```
##  [1] "accel_belt_z"      "roll_belt"         "accel_belt_y"
##  [4] "total_accel_belt"  "accel_dumbbell_z"  "accel_belt_x"
##  [7] "pitch_belt"        "magnet_dumbbell_x" "accel_dumbbell_y"
## [10] "magnet_dumbbell_y" "accel_dumbbell_x"  "accel_arm_x"
## [13] "accel_arm_z"       "magnet_arm_y"      "magnet_belt_z"
## [16] "accel_forearm_y"   "gyros_forearm_y"   "gyros_dumbbell_x"
## [19] "gyros_dumbbell_z"  "gyros_arm_x"
```

Build a Model

This project will build the models with classification trees and random forests to predict the outcome.

Following are three diffrent ways used to predict * Classification Tree * Random Forest * Generalized Boosted Model

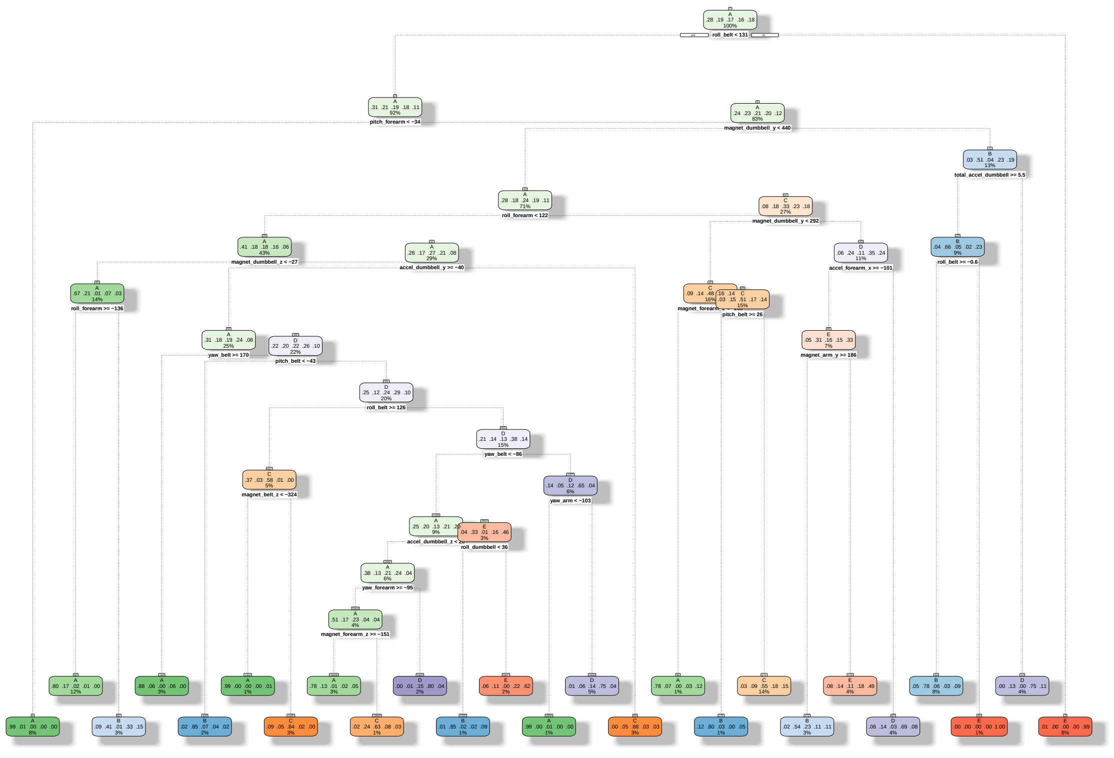
Classification Tree

Create Model

- Obtain model using rpart
- use fancyRpartPlot to plot the classification tree as a dendogram

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

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Rattle 2020−Apr−06 22:28:43 chandani

Prediction

validate the “decisionTreeModel” on testData to find accuracy

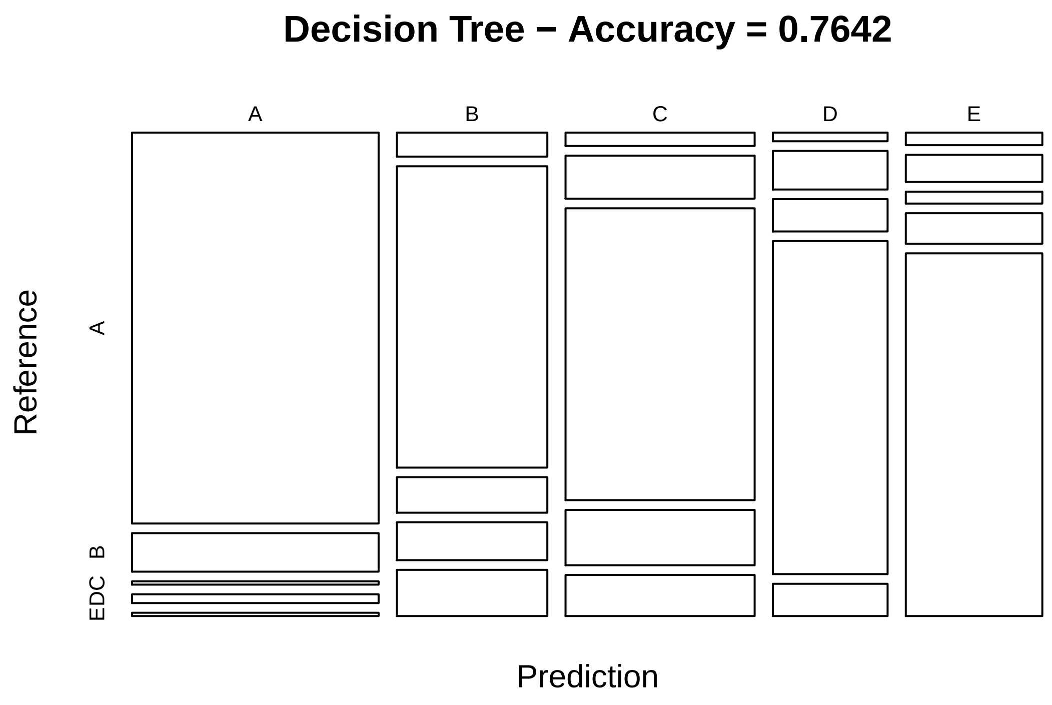
```
predicttionDecisionTreeModel <- predict(decisionTreeModel, testData, type = "class")
cmtree <- confusionMatrix(predicttionDecisionTreeModel, testData$classe)
cmtree
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1067  105    9   24    9
##          B   40  502   59   63   77
##          C   28   90  611  116   86
##          D   11   49   41  423   41
##          E   19   41   18   46  548
##
## Overall Statistics
##
##                Accuracy : 0.7642
##                  95% CI : (0.751, 0.7771)
##     No Information Rate : 0.2826
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7015
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9159   0.6379   0.8279   0.6295   0.7201
## Specificity            0.9503   0.9284   0.9055   0.9589   0.9631
## Pos Pred Value         0.8789   0.6775   0.6563   0.7487   0.8155
## Neg Pred Value         0.9663   0.9157   0.9602   0.9300   0.9383
## Prevalence             0.2826   0.1909   0.1790   0.1630   0.1846
## Detection Rate         0.2588   0.1218   0.1482   0.1026   0.1329
## Detection Prevalence   0.2944   0.1797   0.2258   0.1370   0.1630
## Balanced Accuracy      0.9331   0.7831   0.8667   0.7942   0.8416
```

Accuracy Rate: *0.7642* which is low beacuse of this out-of-sample-error is about *0.23* which is considerable

Plot matrix Results

```
plot(cmtree$table, col = cmtree$byClass,
     main = paste("Decision Tree - Accuracy =", round(cmtree$overall['Accuracy'], 4)))
```



Random Forest

Create Model

- create trainControl
- Train the Random Forest Model

```
# Prediction with Random Forest Tree
controlRandomForest <- trainControl(method="cv", number=3, verboseIter=FALSE)
randomForestModel <- train(classe ~ ., data=trainData, method="rf", trControl=controlRandomForest)
randomForestModel$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: 0.7%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3902    3    0    0    1 0.001024066
## B   19 2634    5    0    0 0.009029345
## C    0   17 2369   10    0 0.011268781
## D    0    1   26 2224    1 0.012433393
## E    0    2    5    6 2512 0.005148515
```

Prediction

validate the "randomForestModel" on testData to find accuracy

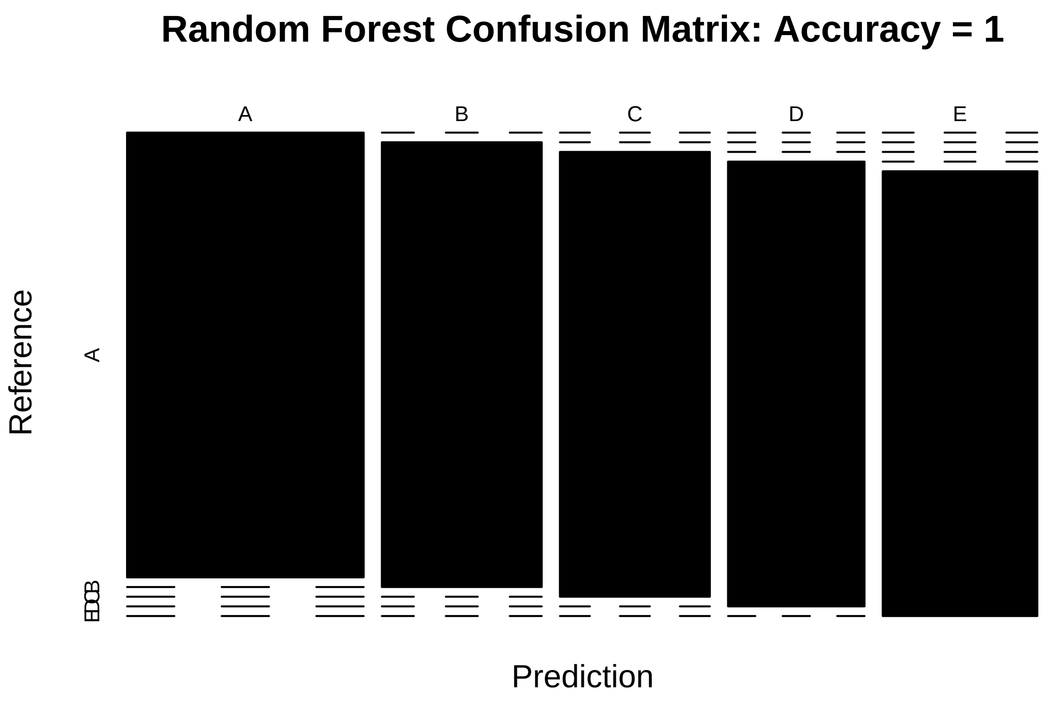
```
predictRandomForestModel <- predict(randomForestModel, newdata=testData)
cmRandomForestModel <- confusionMatrix(predictRandomForestModel, testData$classe)
cmRandomForestModel
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1165    0    0    0    0
##          B    0  787    0    0    0
##          C    0    0  738    0    0
##          D    0    0    0  672    0
##          E    0    0    0    0  761
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9991, 1)
##     No Information Rate : 0.2826
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000    1.000    1.000   1.0000
## Specificity            1.0000   1.0000    1.000    1.000   1.0000
## Pos Pred Value         1.0000   1.0000    1.000    1.000   1.0000
## Neg Pred Value         1.0000   1.0000    1.000    1.000   1.0000
## Prevalence             0.2826   0.1909    0.179    0.163   0.1846
## Detection Rate         0.2826   0.1909    0.179    0.163   0.1846
## Detection Prevalence   0.2826   0.1909    0.179    0.163   0.1846
## Balanced Accuracy      1.0000   1.0000    1.000    1.000   1.0000
```

Accuracy Rate: *1* which is very high so out-of-sample-error is *0*. This might be case of overfitting

Plot matrix Results

```
plot(cmRandomForestModel$table, col = cmRandomForestModel$byClass, main = paste("Random Forest Confusio
```



Generalized Boosted Model

Create Model

- create trainControl
- Train the Generalized Boosted Model

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

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

```
print(modelGBM)
```

```
## Stochastic Gradient Boosting
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 10990, 10990, 10989, 10991, 10988
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7521285  0.6858434
##   1                  100      0.8227397  0.7756753
##   1                  150      0.8522224  0.8130469
##   2                   50      0.8564452  0.8181267
##   2                  100      0.9059465  0.8809760
##   2                  150      0.9301168  0.9115592
##   3                   50      0.8969931  0.8695557
##   3                  100      0.9398712  0.9238994
##   3                  150      0.9588707  0.9479567
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## 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, shrinkage = 0.1 and n.minobsinnode = 10.
```

Prediction

validate the "modelGBM" on testData to find accuracy

```
predictGBM <- predict(modelGBM, newdata=testData)
cmGBM <- confusionMatrix(predictGBM, testData$classe)
cmGBM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1155   20    0    0    1
##          B    9  754   17    5    6
##          C    1   12  713   16    3
##          D    0    1    6  647    8
##          E    0    0    2    4  743
##
## Overall Statistics
##
##                Accuracy : 0.9731
##                  95% CI : (0.9677, 0.9778)
##     No Information Rate : 0.2826
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.966
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9914   0.9581   0.9661   0.9628   0.9763
## Specificity           0.9929   0.9889   0.9905   0.9957   0.9982
## Pos Pred Value        0.9821   0.9532   0.9570   0.9773   0.9920
## Neg Pred Value        0.9966   0.9901   0.9926   0.9928   0.9947
## Prevalence            0.2826   0.1909   0.1790   0.1630   0.1846
## Detection Rate        0.2801   0.1829   0.1729   0.1569   0.1802
## Detection Prevalence  0.2852   0.1919   0.1807   0.1606   0.1817
## Balanced Accuracy     0.9922   0.9735   0.9783   0.9792   0.9873
```

Accuracy Rate: *0.9731* which is very high so out-of-sample-error is *0.06*. This might be case of overfitting

Applying the validation data to Best Model

The accuracy of the 3 regression modeling methods above are:

Random Forest : 1 Decision Tree : 0.7368 GBM : 0.97

Apply the 20 validation results to randomforest model

```r
results <- predict(randomForestModel, newdata=validData)
results
```

```
##  [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
```