Practical Machine Learning Course Project

Muhammad Farhan Mirza

September 16, 2018

This document describes the implementation of Practical Machine Learning Course Project.

Project Description & Intended Results

"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 (http://groupware.les.inf.puc-rio.br/har) (see the section on the Weight Lifting Exercise Dataset)."

The goal of this project is to predict the manner in which they did the exercise. This is the ??? classe??? variable in the training set. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

1. Your submission should consist of a link to a Github repo with your R markdown and compiled HTML file describing your analysis. Please constrain the text of the writeup to < 2000 words and the number of figures to be less than 5. It will make it easier for the graders if you submit a repo with a gh-pages branch so the HTML page can be viewed online (and you always want to make it easy on graders :-).
2. You should also apply your machine learning algorithm to the 20 test cases available in the test data above. Please submit your predictions in appropriate format to the programming assignment for automated grading. See the programming assignment for additional details.

Include Dependant Library Directories

```
library(caret)
```

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

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

```
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 3.5.1
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

Getting and Cleaning Data

Load the training and test data directly from web links into R.

```
trainURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
trainFileName <- "pml-training.csv"
if ( !file.exists(trainFileName) ){
  download.file(url=trainURL, destfile = trainFileName, method = "curl")
}

testURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
testFileName <- "pml-testing.csv"
if ( !file.exists(testFileName) ){
  download.file(url = testURL, destfile = testFileName, method = "curl")
}

pmlTraining <- read.csv(trainFileName, header = TRUE)
pmlTesting <- read.csv(testFileName, header = TRUE)
```

There are 160 variables in the training and testing datasets, which also includes NA values, Identification columns and also some variables have near zero variance which needs to be removed first in order to train the model.

First we will remove the variables which have mostly NA values.

```
NAVars <- sapply(pmlTraining, function(x) mean(is.na(x))) < 0.95
pmlTraining <- pmlTraining[,NAVars]
```

Now, lets remove the variables having variance near zero.

```
nzv <- nearZeroVar(pmlTraining)
pmlTraining <- pmlTraining[,-nzv]
```

Now, remove the identification columns from training dataset.

```
pmlTraining <- pmlTraining[,-(1:5)]
```

Now partition the data into training and test set with the ratio of 60:40, so that we can test our model without applying on the evaluation data available in “pml-testing.csv”.

```
indexTrain <- createDataPartition(pmlTraining$classe, p = 0.6, list = FALSE)
trainSet <- pmlTraining[indexTrain,]
testSet <- pmlTraining[-indexTrain,]
```

Prediction - Model Selection

Now I will dive into training the model and apply the different models on the ***testSet***, so that we can check the accuracy for better model selection to check the model for ***pml-testing***.

1. Gradiant Boosted Model

Train the GBM model and apply prediction on ***testSet***.

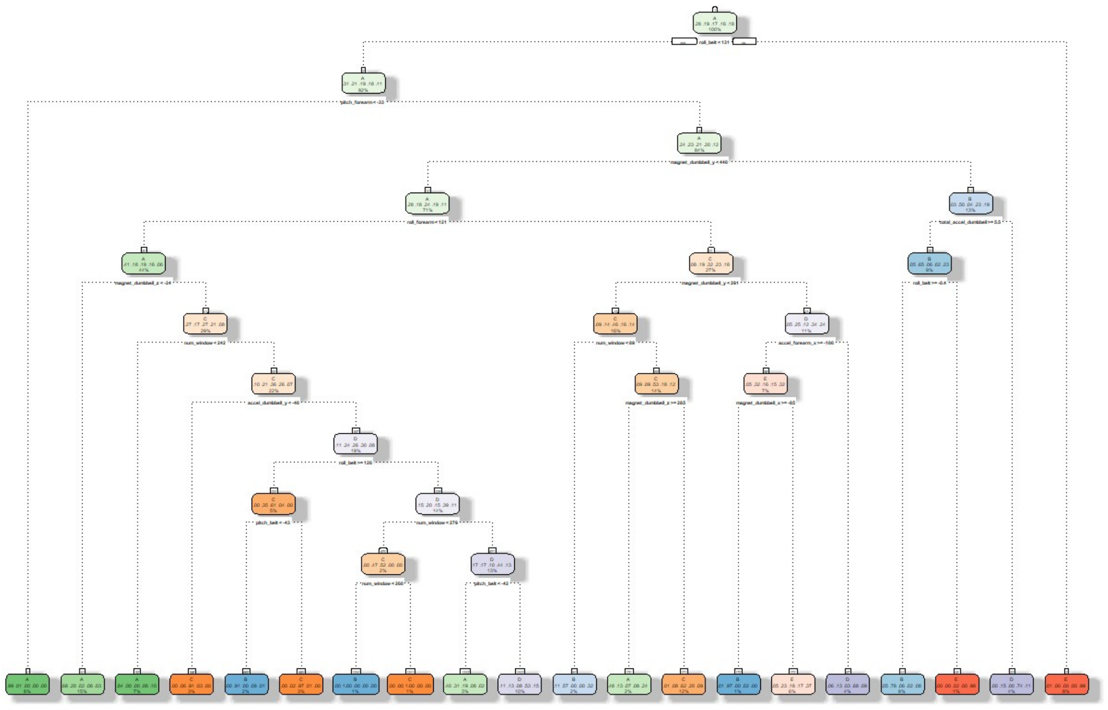
```
set.seed(1111)
gbmFit <- train(classe~., data = trainSet, method = "gbm", verbose = FALSE)
gbmPredict <- predict(gbmFit, newdata = testSet)
confusionMatrix(gbmPredict, testSet$classe)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 2226   11    0    0    1
##          B    5 1493   17    3    9
##          C    0   14 1348   13    4
##          D    1    0    2 1264   10
##          E    0    0    1    6 1418
## 
## Overall Statistics
## 
##                Accuracy : 0.9876
##                  95% CI : (0.9849, 0.99)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
## 
##                   Kappa : 0.9844
##  Mcnemar's Test P-Value : NA
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9973   0.9835   0.9854   0.9829   0.9834
## Specificity            0.9979   0.9946   0.9952   0.9980   0.9989
## Pos Pred Value         0.9946   0.9777   0.9775   0.9898   0.9951
## Neg Pred Value         0.9989   0.9960   0.9969   0.9967   0.9963
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2837   0.1903   0.1718   0.1611   0.1807
## Detection Prevalence   0.2852   0.1946   0.1758   0.1628   0.1816
## Balanced Accuracy      0.9976   0.9891   0.9903   0.9905   0.9911
```

2. Classification Trees

Train the Classification Trees model and apply prediction on ***testSet***.

```
set.seed(1111)
#ctFit <- train(classe~., data = trainSet, method = "rpart")
ctFit <- rpart::rpart(classe~., data = trainSet, method = "class")
fancyRpartPlot(ctFit)
```



Rattle 2018-Sep-18 17:28:07 farha

```
ctPredict <- predict(ctFit, newdata = testSet, type = "class")
confusionMatrix(ctPredict, testSet$classe)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 2024  359   61  174  115
##          B   51  818   40   29  115
##          C   12   75 1095  188   85
##          D  112  158   76  817  153
##          E   33  108   96   78  974
## 
## Overall Statistics
## 
##                Accuracy : 0.7301          
##                  95% CI : (0.7201, 0.7399)
##     No Information Rate : 0.2845          
##     P-Value [Acc > NIR] : < 2.2e-16       
## 
##                   Kappa : 0.6561          
##  Mcnemar's Test P-Value : < 2.2e-16       
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9068   0.5389   0.8004   0.6353   0.6755
## Specificity            0.8737   0.9629   0.9444   0.9239   0.9508
## Pos Pred Value         0.7406   0.7768   0.7526   0.6208   0.7556
## Neg Pred Value         0.9593   0.8970   0.9573   0.9282   0.9286
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2580   0.1043   0.1396   0.1041   0.1241
## Detection Prevalence   0.3483   0.1342   0.1854   0.1677   0.1643
## Balanced Accuracy      0.8903   0.7509   0.8724   0.7796   0.8131
```

3. Random Forest

Train the Random Forest model and apply prediction on ***testSet***.

```
set.seed(1111)
rfFit <- train(classe~., data = trainSet, method = "rf")
rfPredict <- predict(rfFit, newdata = testSet)
confusionMatrix(rfPredict, testSet$classe)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 2231    4    0    0    0
##          B    0 1512    3    0    0
##          C    0    2 1365    5    0
##          D    0    0    0 1281    2
##          E    1    0    0    0 1440
## 
## Overall Statistics
## 
##                Accuracy : 0.9978
##                  95% CI : (0.9965, 0.9987)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
## 
##                   Kappa : 0.9973
##  Mcnemar's Test P-Value : NA
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9996   0.9960   0.9978   0.9961   0.9986
## Specificity            0.9993   0.9995   0.9989   0.9997   0.9998
## Pos Pred Value         0.9982   0.9980   0.9949   0.9984   0.9993
## Neg Pred Value         0.9998   0.9991   0.9995   0.9992   0.9997
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1927   0.1740   0.1633   0.1835
## Detection Prevalence   0.2849   0.1931   0.1749   0.1635   0.1837
## Balanced Accuracy      0.9994   0.9978   0.9984   0.9979   0.9992
```

4. Linear Discriminant Analysis

Train the Linear Discriminant Analysis model and apply prediction on ***testSet***.

```
set.seed(1111)
ldaFit <- train(classe~., data = trainSet, method = "lda")
ldaPredict <- predict(ldaFit, newdata = testSet)
confusionMatrix(ldaPredict, testSet$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1842  203  160   57   59
##          B   64  972  135   50  194
##          C  160  205  876  190  137
##          D  151   71  164  949  145
##          E   15   67   33   40  907
##
## Overall Statistics
##
##                Accuracy : 0.7069
##                  95% CI : (0.6966, 0.7169)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6291
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8253   0.6403   0.6404   0.7379   0.6290
## Specificity            0.9147   0.9300   0.8932   0.9191   0.9758
## Pos Pred Value         0.7936   0.6869   0.5587   0.6412   0.8540
## Neg Pred Value         0.9294   0.9151   0.9216   0.9471   0.9211
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2348   0.1239   0.1116   0.1210   0.1156
## Detection Prevalence   0.2958   0.1803   0.1998   0.1886   0.1354
## Balanced Accuracy      0.8700   0.7852   0.7668   0.8285   0.8024
```

Out Of Sample Error

```
oose.lda <- 1 - ( sum(ldaPredict == testSet$classe) / length(ldaPredict) )
oose.gbm <- 1 - ( sum(gbmPredict == testSet$classe) / length(gbmPredict) )
oose.rf <- 1 - ( sum(rfPredict == testSet$classe) / length(rfPredict) )
oose.ct <- 1 - ( sum(ctPredict == testSet$classe) / length(ctPredict) )
```

The Out of Sample Error for

1. Linear Discriminant Analysis is **0.293143**
2. Gradiant Boosted Model is **0.012363**
3. Random Forest is **0.0021667**
4. Classification Trees is **0.2699465**

Accuracy

The accuracy of the models discussed above is

1. Linear Discriminant Analysis is **70.69%**
2. Gradiant Boosted Model is **98.76%**
3. Random Forest is **99.78%**
4. Classification Trees is **73.01%**

Predicting quiz results

As we can see that the Random Forest model has the highest accuracy **99.78%** as well as lowest Out of sample error is **0.216%**. So, I am selecting Random Forest for the prediction of ***pml-testing*** dataset for the quiz results.

```
predPMLTesting <- predict(rfFit, newdata = pmlTesting)
predPMLTesting
```

```
##  [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
```