Johns Hopkins Data Science Project (Coursera) : Practical Machine Learning

Predicting Exercise Activity Quality

By Chetan Bhatt

Synopsis

Using devices such as JawboneUp, NikeFuelBand and Fitbit, it is now possible to collect a large amount of data about personal activity relatively inexpensively. In this project we will use data from the study 'Qualitative Activity Recognition of Weight Lifting Exercises' (see reference), in which exercise data was collected for six participants participated in a dumbell lifting exercise five different ways, using such electronic devices through their sensors. Our goal here is to apply appropriate machine learning algorithm, training and testing data and predict the manner in which they did the exercise. Question here is how best can we predict the quality of the exercise activity?

GIT REPO For the Project is HERE : **https://github.com/xde0037/Practical_Machine_Learning.git**

Sourcing, Loading and Processing Data

We will use the training and testing data from the study as mentioned in this report: [1] Training Data : http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv [2] Testing Data : http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

```r
# load essential libraries
suppressMessages(library(caret))
suppressMessages(library(ggplot2))
suppressMessages(library(randomForest))
suppressMessages(library(AppliedPredictiveModeling))
suppressMessages(library(rattle))
suppressMessages(library(rpart.plot))
suppressMessages(library(gbm))

# download the data files if they do not already exist.

rm(list = ls())
if (!file.exists("pml-training.csv")) {
  download.file("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv", destfile = "pml-
}
if (!file.exists("pml-testing.csv")) {
  download.file("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv", destfile = "pml-te
}
# read testing and training data from CSV file and define NA's, nulls and error rows.
TestingData  <- read.csv("pml-testing.csv",  sep = ",", na.strings = c("", "NA", "#DIV/0!"))
TrainingData <- read.csv("pml-training.csv", sep = ",", na.strings = c("", "NA", "#DIV/0!"))

# Column Names
TestingColNames <- colnames(TestingData)
TrainingColNames <- colnames(TrainingData)
# this checks column names in both set are exactly same (excluding classe and problem_id).
all.equal(TrainingColNames[1:length(TrainingColNames)-1], TestingColNames[1:length(TestingColNames)-1])
```

```
## [1] TRUE
```

– We will apply all analysis on **TrainingData**. – The **str(TrainingData)** command shows that we have **19622** observations, and **160** variables. – From dataset, we can see that first 7 columns are not required and most important variable is **classe**, which we want to predict, while using other variables.

Partitioning data for test and training:

We will partition data into 60% and 40% for training set and testing set respectively.

```r
set.seed(39)
Train_Partition <- createDataPartition(y=TrainingData$classe, p=0.6, list=FALSE)
TrainPart <- TrainingData[Train_Partition, ]
TestPart  <- TrainingData[-Train_Partition, ]
```

Cleaning Data

– We will remove all NA type variables

```r
# Total non-NAs in each column
nonNAs <- function(x) {
    as.vector(apply(x, 2, function(x) length(which(!is.na(x)))))
}
# Get the vector of NA columns to drop.
NumCols <- nonNAs(TrainPart)
dropCols <- c()
for (I in 1:length(NumCols)) {
    if (NumCols[I] < nrow(TrainPart)) {
        dropCols <- c(dropCols, TrainingColNames[I])
    }
}

# TidyUp Training Data : First 7 columns are not required for prediction so we remove them alongwith al
TrainingPart <- TrainPart[,!(names(TrainPart) %in% dropCols)]
TrainingPart <- TrainingPart[,8:length(colnames(TrainingPart))]

# Similarly remove same set of data and columns for testing partition as well
TestingPart <- TestPart[,!(names(TestPart) %in% dropCols)]
TestingPart <- TestingPart[,8:length(colnames(TestingPart))]

# Similarly for the Submission Data Set as got from testing data csv file from download.
SubmitPart_01 <- TestingData[,!(names(TestingData) %in% dropCols)]
SubmitPart    <- SubmitPart_01[,8:length(colnames(SubmitPart_01))]
Submission_Test_Data <- SubmitPart
```

– We will remove all variable with near zero variance:

```r
nzv <- nearZeroVar(TrainingPart, saveMetrics=TRUE)
TrainingNZV <- TrainingPart[,nzv$nzv==FALSE]

nzv <- nearZeroVar(TestingPart,saveMetrics=TRUE)
TestingNZV <- TestingPart[,nzv$nzv==FALSE]

TrainingPart <- TrainingNZV
TestingPart  <- TestingNZV
```

```r
# training set will be used in model building
dim(TrainingPart)
```

```
## [1] 11776    53
```

```r
# testing set will be used in applying the algorithms
dim(TestingPart)
```

```
## [1] 7846   53
```

```r
# this data set is for submission of final prediction output
dim(Submission_Test_Data)
```

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

Algorithms And Model Selection

In order to arrive at decision sooner, we will use testing data to apply ML algorithms. We will use three algorithms (based on information from the reference paper) here on testing data : Classification Tree **(rpart)** and Random Forest **(rf)** and Gradient Boosting **(gbm)**. (Reference to lecture notes).

We will then compare accuracy of each model to see which one provides best accuracy. These steps are provided below:

[1] Classification Tree :

Let us look at decision tree algorithm and its prediction.

```r
set.seed(12345)
# Create Tree Model with rpart function, With Training Data Set
CtModFit <- rpart(classe ~ ., data=TrainingPart, method="class")
```

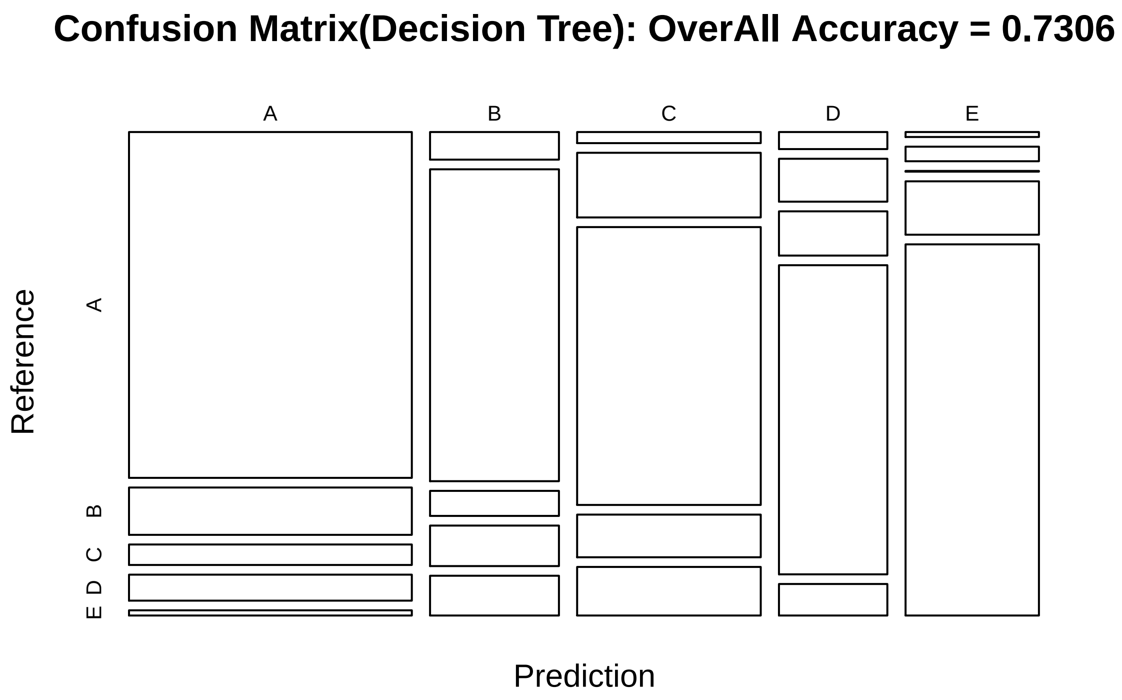
Prediction and Accuracy for Decision Tree:

```r
# Generate prediction stats for Decision Tree and then Confusion Matrix, using **Testing** Data Set.
CtPrediction <- predict(CtModFit, TestingPart, type = "class")
CT_CMatrix <- confusionMatrix(CtPrediction, TestingPart$classe)
CT_CMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2061  282  122  155   31
##          B   75  847   68  110  108
##          C   43  250 1075  165  188
##          D   39   98  101  706   72
##          E   14   41    2  150 1043
##
## Overall Statistics
```

```
##
##                Accuracy : 0.7306
##                  95% CI : (0.7206, 0.7404)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6569
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9234   0.5580   0.7858  0.54899   0.7233
## Specificity            0.8949   0.9430   0.9003  0.95274   0.9677
## Pos Pred Value         0.7774   0.7012   0.6246  0.69488   0.8344
## Neg Pred Value         0.9671   0.8989   0.9522  0.91508   0.9395
## Prevalence             0.2845   0.1935   0.1744  0.16391   0.1838
## Detection Rate         0.2627   0.1080   0.1370  0.08998   0.1329
## Detection Prevalence   0.3379   0.1540   0.2193  0.12949   0.1593
## Balanced Accuracy      0.9091   0.7505   0.8430  0.75087   0.8455
```

```
# View the Confusion matrix plot for Decision Tree
plot(CT_CMatrix$table, col = CT_CMatrix$byClass, main = paste("Confusion Matrix(Decision Tree): OverAll
                        round(CT_CMatrix$overall['Accuracy'], 4)))
```



[2] Random Forest Model :

```
Now let us look at Random Forest Algorithm :
```

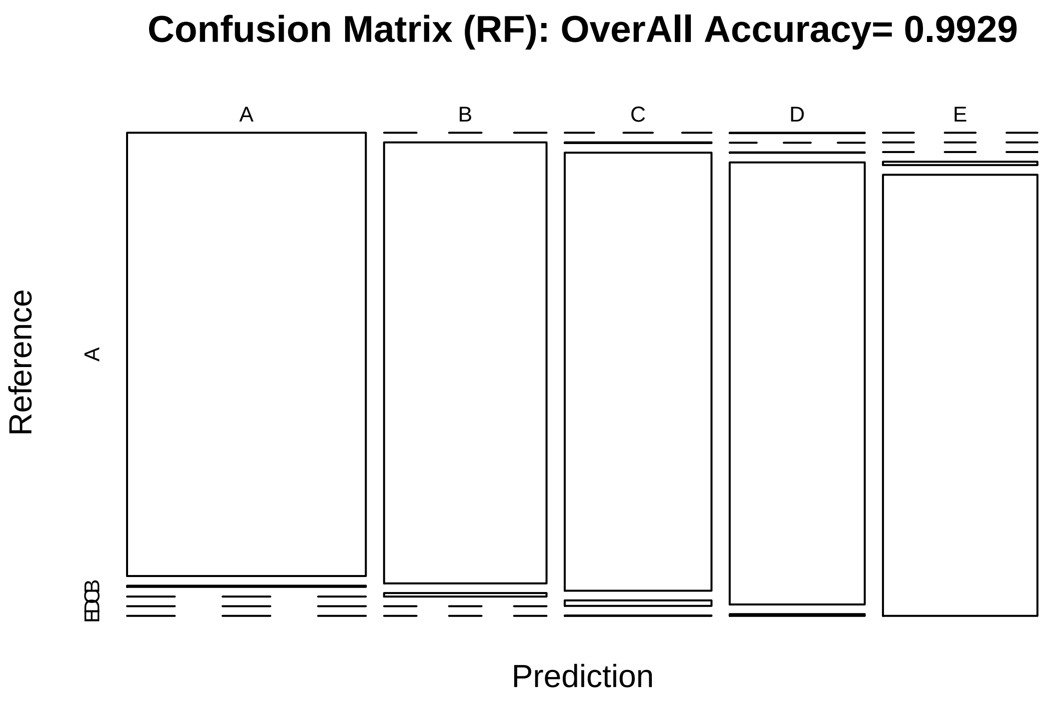
```
set.seed(12345)
# Apply randomForest ML algorithm and create Random Forest Model, With Training Data Set
RfModFit <- randomForest(classe ~ ., data=TrainingPart)
RF_TrainFit <- train(classe ~ ., method = "rf",
                data = TrainingPart, importance = T,
                trControl = trainControl(method = "cv", number = 3))
```

Prediction and Accuracy for Random Forest Algorithm:

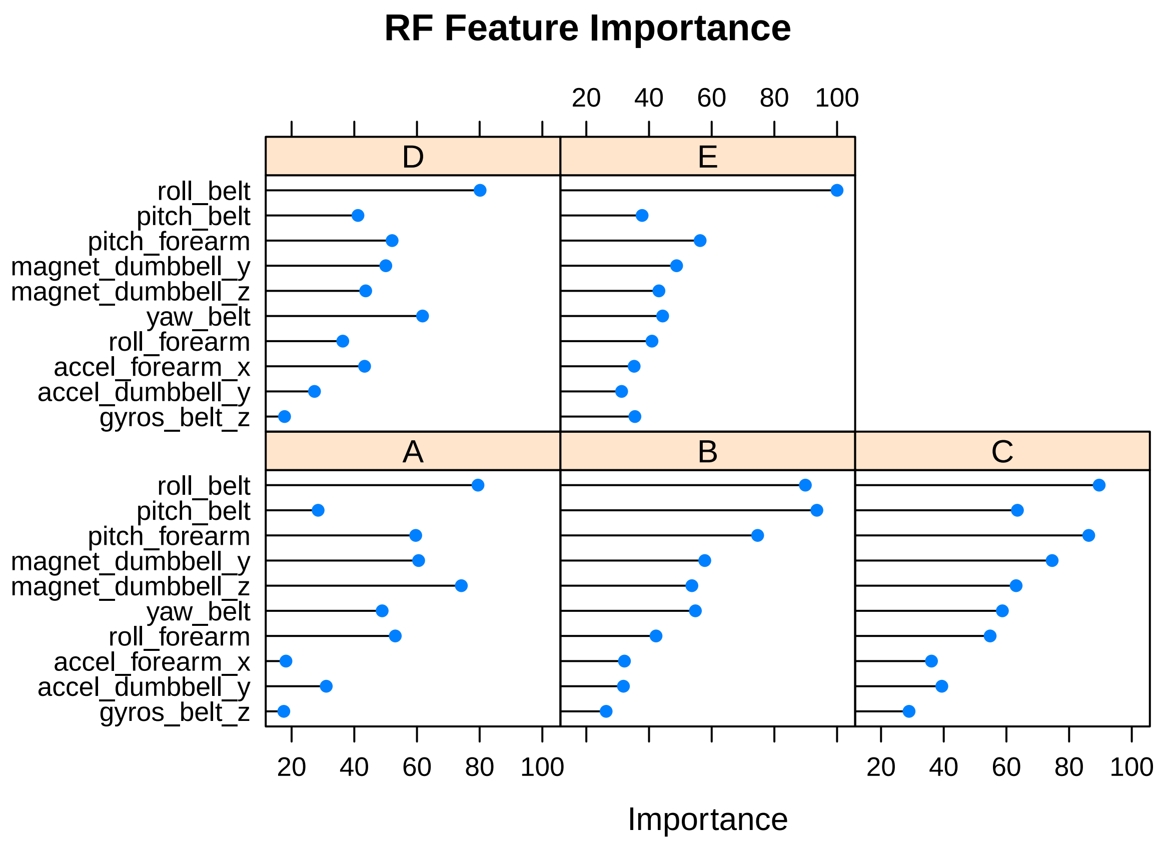
```
# Generate prediction stats for RF Algorithm and then Confusion Matrix, using **Testing** Data Set.
RF_prediction <- predict(RfModFit, TestingPart, type = "class")
RF_CMatrix<- confusionMatrix(RF_prediction, TestingPart$classe)
RF_CMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2231    6    0    0    0
##          B    0 1510   12    0    0
##          C    0    2 1355   17    1
##          D    1    0    1 1258    5
##          E    0    0    0   11 1436
##
## Overall Statistics
##
##                Accuracy : 0.9929
##                  95% CI : (0.9907, 0.9946)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.991
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9996   0.9947   0.9905   0.9782   0.9958
## Specificity            0.9989   0.9981   0.9969   0.9989   0.9983
## Pos Pred Value         0.9973   0.9921   0.9855   0.9945   0.9924
## Neg Pred Value         0.9998   0.9987   0.9980   0.9957   0.9991
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1925   0.1727   0.1603   0.1830
## Detection Prevalence   0.2851   0.1940   0.1752   0.1612   0.1844
## Balanced Accuracy      0.9992   0.9964   0.9937   0.9886   0.9971
```

```r
# View the Confusion matrix plot for Random Forest
plot(RF_CMatrix$table, col = RF_CMatrix$byClass, main = paste("Confusion Matrix (RF): OverAll Accuracy=
                 round(RF_CMatrix$overall['Accuracy'], 4)))
```



```r
plot(varImp(RF_TrainFit), top = 10, main="RF Feature Importance")
```



[3] Model and Prediction Using Gradient Boosting Algorithm (GBM):

```
set.seed(12345)
# keep 5 folds (number of sampling iterations) and 1-fold repeated cross validations
T_Control <- trainControl(method = "repeatedcv",number = 5,repeats = 1)

gbmModelFit <- train(classe ~ ., data=TrainingPart, method = "gbm",trControl = T_Control,verbose=FALSE)
```

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

```
gbmFinModel <- gbmModelFit$finalModel

gbmPrediction    <- predict(gbmModelFit, newdata=TestingPart)
gbmAccuracyStats <- confusionMatrix(gbmPrediction, TestingPart$classe)
gbmAccuracyStats
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction     A    B    C    D    E
##            A 2195   57    0    0    7
##            B   24 1433   41    4   16
##            C    5   27 1316   52   14
##            D    6    0    9 1222   17
##          E    2    1    2    8 1388
##
## Overall Statistics
##
##                Accuracy : 0.9628
##                  95% CI : (0.9584, 0.9669)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9529
##  Mcnemar's Test P-Value : 5.628e-15
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9834   0.9440   0.9620   0.9502   0.9626
## Specificity            0.9886   0.9866   0.9849   0.9951   0.9980
## Pos Pred Value         0.9717   0.9440   0.9307   0.9745   0.9907
## Neg Pred Value         0.9934   0.9866   0.9919   0.9903   0.9916
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2798   0.1826   0.1677   0.1557   0.1769
## Detection Prevalence   0.2879   0.1935   0.1802   0.1598   0.1786
## Balanced Accuracy      0.9860   0.9653   0.9734   0.9727   0.9803
```

Results

Random Forest Algorithm provided the best accuracy for testing data.

Comparing the overall accuracy as observed for Decision Tree (**73.06%**), the Random Forest Algorithm Accuracy (**99.29%**) And the Accuracy as seen in GBM (**96.28%**), We can conclude that Random Forest provides the best accuracy over other algorithms. The **Out-Of-Sample** error is (**100.00-99.29=0.71**). So, expected error rate of less than 1% an important aspect fulfilled.

– For Random Forest, we will finally apply prediction to the in-sample testing data. We have been given 20 test cases to generate predictions for them, we will then store the results in text files for each case.

```
Final_RF_Prediction <- predict(RfModFit, Submission_Test_Data, type = "class")
Final_RF_Prediction
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```

We will now use **Final__RF__Prediction** to generate prediction data files. It generates 20 output files.

```
Gen_PML_Files = function(x){
  n = length(x)
  for(i in 1:n){
    filename = paste0("problem_id_",i,".txt")
    write.table(x[i],file=filename,quote=FALSE,row.names=FALSE,col.names=FALSE)
  }
}

Gen_PML_Files(Final_RF_Prediction)
```

References :

[1] Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13). Stuttgart, Germany: ACM SIGCHI, 2013.

[2] Lecture Videos and Notes from Practical Machine Learning Course , JHU, Coursera.