

practical machine learning week 4 project solution -Million Nzvuwu

Million Nzvuwu

28 December 2018

Final Project Report - Practical Machine Learning Course

Introduction :

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 dumbbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways.

Data

The training data for this project are available here:

<https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv>
(<https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv>)

The test data are available here:

<https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv>
(<https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv>)

Initialization

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.4.4
```

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

```
## Warning: package 'lattice' was built under R version 3.4.4
```

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

```
## Warning: package 'ggplot2' was built under R version 3.4.4
```

```
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 3.4.4
```

```
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 3.4.4
```

```
library(RColorBrewer)
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 3.4.4
```

```
## 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.
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.4.4
```

```
## randomForest 4.6-14
```

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

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:rattle':
##
##     importance
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

Finally, load the same seed with the following line of code:

```
set.seed(12345)
```

Getting the data

Load data to memory solely

```
training <- read.csv('pml_training.csv', na.strings=c("NA", "#DIV/0!", ""))  
testing <- read.csv('pml_testing.csv', na.strings=c("NA", "#DIV/0!", ""))
```

Partitioning the training set into two

Partitioning Training data set into two data sets, 60% for myTraining, 40% for myTesting:

```
inTrain <- createDataPartition(y=training$classe, p=0.6, list=FALSE)  
myTraining <- training[inTrain, ]; myTesting <- training[-inTrain, ]  
dim(myTraining); dim(myTesting)
```

```
## [1] 11776 160
```

```
## [1] 7846 160
```

Cleaning the data

The following transformations were used to clean the data:

Transformation 1: Cleaning NearZeroVariance Variables Run this code to view possible NZV Variables:

```
myDataNZV <- nearZeroVar(myTraining, saveMetrics=TRUE)
```

Run this code to create another subset without NZV variables:

```
myNZVvars <- names(myTraining) %in% c("new_window", "kurtosis_roll_belt", "kurtosis_pitch_belt",
"kurtosis_yaw_belt", "skewness_roll_belt", "skewness_roll_belt.1", "skewness_yaw_belt",
"max_yaw_belt", "min_yaw_belt", "amplitude_yaw_belt", "avg_roll_arm", "stddev_roll_arm",
"var_roll_arm", "avg_pitch_arm", "stddev_pitch_arm", "var_pitch_arm", "avg_yaw_arm",
"stddev_yaw_arm", "var_yaw_arm", "kurtosis_roll_arm", "kurtosis_pitch_arm",
"kurtosis_yaw_arm", "skewness_roll_arm", "skewness_pitch_arm", "skewness_yaw_arm",
"max_roll_arm", "min_roll_arm", "min_pitch_arm", "amplitude_roll_arm", "amplitude_pitch_arm",
"kurtosis_roll_dumbbell", "kurtosis_pitch_dumbbell", "kurtosis_yaw_dumbbell", "skewness_roll_dumbbell",
"skewness_pitch_dumbbell", "skewness_yaw_dumbbell", "max_yaw_dumbbell", "min_yaw_dumbbell",
"amplitude_yaw_dumbbell", "kurtosis_roll_forearm", "kurtosis_pitch_forearm", "kurtosis_yaw_forearm",
"skewness_roll_forearm", "skewness_pitch_forearm", "skewness_yaw_forearm",
"max_roll_forearm",
"max_yaw_forearm", "min_roll_forearm", "min_yaw_forearm", "amplitude_roll_forearm",
"amplitude_yaw_forearm", "avg_roll_forearm", "stddev_roll_forearm", "var_roll_forearm",
"avg_pitch_forearm", "stddev_pitch_forearm", "var_pitch_forearm", "avg_yaw_forearm",
"stddev_yaw_forearm", "var_yaw_forearm")
myTraining <- myTraining[!myNZVvars]
dim(myTraining)
```

```
## [1] 11776 100
```

Transformation 2: Killing first column of Dataset - ID
Removing first ID variable so that it does not interfere with ML Algorithms:

```
myTraining <- myTraining[c(-1)]
```

Transformation 3: Cleaning Variables with too many NAs. For Variables that have more than a 60% threshold of NA's I'm going to leave them out:

```
trainingV3 <- myTraining #creating another subset to iterate in loop
for(i in 1:length(myTraining)) { #for every column in the training dataset
  if( sum( is.na( myTraining[, i] ) ) /nrow(myTraining) >= .6 ) { #if
n?? NAs > 60% of total observations
    for(j in 1:length(trainingV3)) {
      if( length( grep(names(myTraining[i]), names(trainingV3)[j]) ) =
=1) { #if the columns are the same:
        trainingV3 <- trainingV3[ , -j] #Remove that column
      }
    }
  }
}
#To check the new N?? of observations
dim(trainingV3)
```

```
## [1] 11776    58
```

```
#Seting back to our set:
myTraining <- trainingV3
rm(trainingV3)
```

Now let us do the exact same 3 transformations but for our myTesting and testing data sets.

```
clean1 <- colnames(myTraining)
clean2 <- colnames(myTraining[, -58]) #already with classe column removed
myTesting <- myTesting[clean1]
testing <- testing[clean2]

#To check the new N?? of observations
dim(myTesting)
```

```
## [1] 7846    58
```

```
#To check the new N?? of observations
dim(testing)
```

```
## [1] 20 57
```

Note: The last column - problem_id - which is not equal to

training sets, was also “automagically” removed

No need for this code:

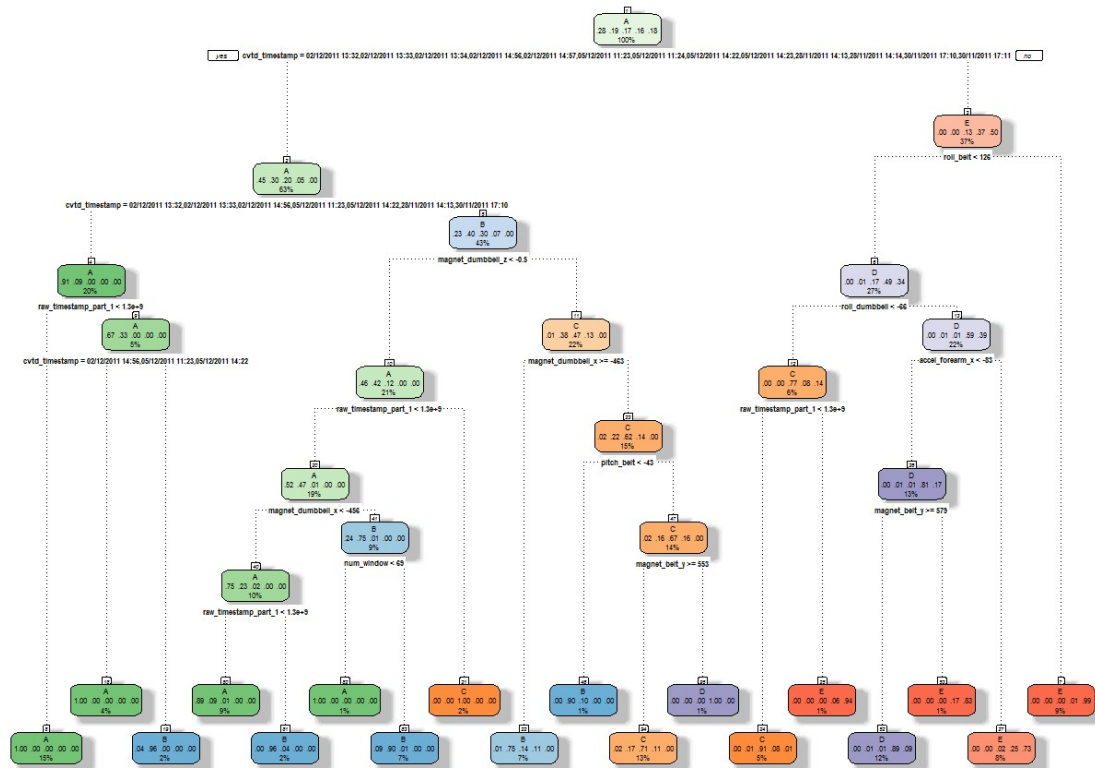
```
testing <- testing[-length(testing)]
```

In order to ensure proper functioning of Decision Trees and especially RandomForest Algorithm with the Test data set (data set provided), we need to coerce the data into the same type.

```
for (i in 1:length(testing) ) {  
  for(j in 1:length(myTraining)) {  
    if( length( grep(names(myTraining[i]), names(testing)[j]) ) ==1)  {  
      class(testing[j]) <- class(myTraining[i])  
    }  
  }  
}  
#And to make sure Coertion really worked, simple smart ass technique:  
testing <- rbind(myTraining[2, -58] , testing) #note row 2 does not mean any  
thing, this will be removed right.. now:  
testing <- testing[-1,]
```

Using ML algorithms for prediction: Decision Tree

```
modFitA1 <- rpart(classe ~ ., data=myTraining, method="class")  
  
##Note: to view the decision tree with fancy run this command:  
fancyRpartPlot(modFitA1)
```



Rattle 2018-Dec-28 15:23:32 Million

```
##plot of chunk unnamed-chunk-17
```

Predicting:

```
predictionsA1 <- predict(modFitA1, myTesting, type = "class")
```

```
##(Moment of truth) Using confusion Matrix to test results:
confusionMatrix(predictionsA1, myTesting$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A    B    C    D    E
##           A 2150   60    7    1    0
##           B   61 1260   69   64    0
##           C   21  188 1269  143    4
##           D    0   10   14  857   78
##           E    0    0    9  221 1360
##
## Overall Statistics
##
##           Accuracy : 0.8789
##           95% CI : (0.8715, 0.8861)
##           No Information Rate : 0.2845
##           P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.8468
##           McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##           Class: A Class: B Class: C Class: D Class: E
## Sensitivity           0.9633   0.8300   0.9276   0.6664   0.9431
## Specificity           0.9879   0.9693   0.9450   0.9845   0.9641
## Pos Pred Value        0.9693   0.8666   0.7809   0.8936   0.8553
## Neg Pred Value        0.9854   0.9596   0.9841   0.9377   0.9869
## Prevalence            0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate        0.2740   0.1606   0.1617   0.1092   0.1733
## Detection Prevalence  0.2827   0.1853   0.2071   0.1222   0.2027
## Balanced Accuracy      0.9756   0.8997   0.9363   0.8254   0.9536
```

Using ML algorithms for prediction: Random Forests

```
modFitB1 <- randomForest(classe ~. , data=myTraining)

##Predicting in-sample error:
predictionsB1 <- predict(modFitB1, myTesting, type = "class")

##(Moment of truth) Using confusion Matrix to test results:
confusionMatrix(predictionsB1, myTesting$classe)
```



```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##           A 2231    2    0    0    0
##           B    1 1516    1    0    0
##           C    0    0 1366    3    0
##           D    0    0    1 1281    1
##           E    0    0    0    2 1441
##
## Overall Statistics
##
##           Accuracy : 0.9986
##           95% CI : (0.9975, 0.9993)
##           No Information Rate : 0.2845
##           P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.9982
##           McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##           Class: A Class: B Class: C Class: D Class: E
## Sensitivity          0.9996    0.9987    0.9985    0.9961    0.9993
## Specificity          0.9996    0.9997    0.9995    0.9997    0.9997
## Pos Pred Value       0.9991    0.9987    0.9978    0.9984    0.9986
## Neg Pred Value       0.9998    0.9997    0.9997    0.9992    0.9998
## Prevalence           0.2845    0.1935    0.1744    0.1639    0.1838
## Detection Rate       0.2843    0.1932    0.1741    0.1633    0.1837
## Detection Prevalence 0.2846    0.1935    0.1745    0.1635    0.1839
## Balanced Accuracy     0.9996    0.9992    0.9990    0.9979    0.9995
```

Conclusion

As we can see from the results, the random forest algorithm far outperforms the decision tree in terms of accuracy. Random Forests yielded better Results, as expected!

Generating Files to submit as answers for the Assignment:

Finally, using the provided Test Set out-of-sample error.

For Random Forests we use the following formula, which yielded a much better prediction in in-sample:

```
predictionsB2 <- predict(modFitB1, testing, type = "class")
#Function to generate files with predictions to submit for assignment
pml_write_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)
  }
}

pml_write_files(predictionsB2)
```