### Install Packages, Libraries and Seed

Installing packages, loading libraries, and setting the seed for reproduceability:

**Install.package(“caret”)**

**library**(caret)

## Warning: package 'caret' was built under R version 3.1.3

## Loading required package: lattice

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 3.1.1

**Install.package(“randomForest”)**

**library**(randomForest)

## Warning: package 'randomForest' was built under R version 3.1.3

## randomForest 4.6-10

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

**Install.package(“rpart”)**

**library**(rpart)

## Warning: package 'rpart' was built under R version 3.1.3

**Install.package(“rpart.plot”)**

**library**(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 3.1.3

**Install.package(“RColorBrewer”)**

**library**(RColorBrewer)

## Warning: package 'RColorBrewer' was built under R version 3.1.1

**Install.package(“rattle”)**

**library**(rattle)

## Warning: package 'rattle' was built under R version 3.1.3

## Rattle: A free graphical interface for data mining with R.

## Version 3.4.1 Copyright (c) 2006-2014 Togaware Pty Ltd.

## Type 'rattle()' to shake, rattle, and roll your data.

set.seed(1234)

**Getting and cleaning data**

The training data set can be found on the following URL:

trainUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"

The testing data set can be found on the following URL:

testUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

Load data to memory.

training <- read.csv(url(trainUrl), na.strings=c("NA","#DIV/0!",""))

testing <- read.csv(url(testUrl), na.strings=c("NA","#DIV/0!",""))

**Partioning the training set into two**

Partioning 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)

Another subset of NZV variables.

myNZVvars <- names(myTraining) %in% c("new\_window", "kurtosis\_roll\_belt", "kurtosis\_picth\_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\_picth\_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\_picth\_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\_picth\_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 interfer 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

*#Setting back to our set:*

myTraining <- trainingV3

rm(trainingV3)

Now let us do the exact same 3 transformations for 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

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 anything, this will be removed right.. now:*

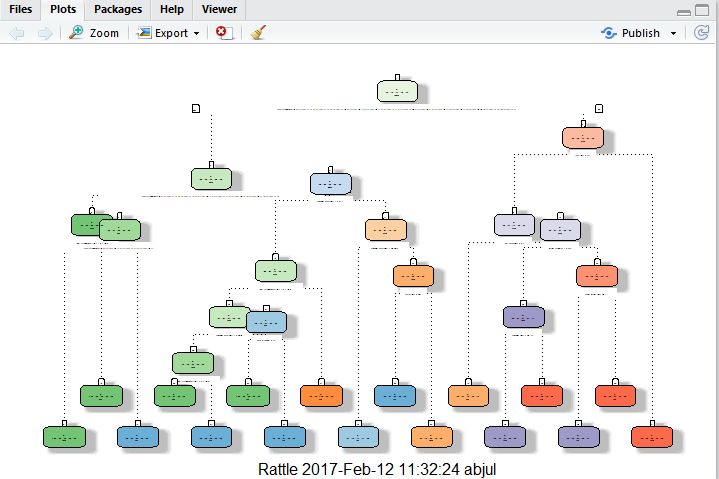
testing <- testing[-1,]

**Using ML algorithms for prediction: Decision Tree**

modFitA1 <- rpart(classe ~ ., data=myTraining, method="class")

To view the decision tree with fancy :

fancyRpartPlot(modFitA1)



**Predicting:**

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

Using confusion Matrix to test results:

confusionMatrix(predictionsA1, myTesting$classe)

## Confusion Matrix and Statistics

##

## Reference

## Prediction A B C D E

## A 2161 61 5 3 0

## B 50 1271 95 64 0

## C 21 177 1242 203 65

## D 0 9 19 899 92

## E 0 0 7 117 1285

##

## Overall Statistics

##

## Accuracy : 0.874

## 95% CI : (0.867, 0.881)

## No Information Rate : 0.284

## P-Value [Acc > NIR] : <2e-16

##

## Kappa : 0.841

## Mcnemar's Test P-Value : NA

##

## Statistics by Class:

##

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

## Sensitivity 0.968 0.837 0.908 0.699 0.891

## Specificity 0.988 0.967 0.928 0.982 0.981

## Pos Pred Value 0.969 0.859 0.727 0.882 0.912

## Neg Pred Value 0.987 0.961 0.979 0.943 0.976

## Prevalence 0.284 0.193 0.174 0.164 0.184

## Detection Rate 0.275 0.162 0.158 0.115 0.164

## Detection Prevalence 0.284 0.189 0.218 0.130 0.180

## Balanced Accuracy 0.978 0.902 0.918 0.840 0.936

**Using ML algorithms for prediction: Random Forests**

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

Predicting in-sample error:

predictionsB1 <- predict(modFitB1, myTesting, type = "class")

Using confusion Matrix to test results:

confusionMatrix(predictionsB1, myTesting$classe)

## Confusion Matrix and Statistics

##

## Reference

## Prediction A B C D E

## A 2232 2 0 0 0

## B 0 1516 4 0 0

## C 0 0 1362 5 0

## D 0 0 2 1280 0

## E 0 0 0 1 1442

##

## Overall Statistics

##

## Accuracy : 0.998

## 95% CI : (0.997, 0.999)

## No Information Rate : 0.284

## P-Value [Acc > NIR] : <2e-16

##

## Kappa : 0.998

## Mcnemar's Test P-Value : NA

##

## Statistics by Class:

##

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

## Sensitivity 1.000 0.999 0.996 0.995 1.000

## Specificity 1.000 0.999 0.999 1.000 1.000

## Pos Pred Value 0.999 0.997 0.996 0.998 0.999

## Neg Pred Value 1.000 1.000 0.999 0.999 1.000

## Prevalence 0.284 0.193 0.174 0.164 0.184

## Detection Rate 0.284 0.193 0.174 0.163 0.184

## Detection Prevalence 0.285 0.194 0.174 0.163 0.184

## Balanced Accuracy 1.000 0.999 0.997 0.998 1.000

Random Forests yielded better Results.

**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)