Practical Machine Learning Take5

Kathy0305

February 4, 2017

Practical Machine Learning Final Project

Background

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).

Data

The data is already divided into two subsets, training and test:

- The training data for this project are available here:Training (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv)
- The test data for this project are available here:Test (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)

The Goal of the Project

The goal of your 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? * Why you made the choices you did? You will also use your prediction model to predict 20 different test cases.

Background about the original data

The reserachers wanted to capture not the quantity (the number of times) of an activity or excercise, but the quality of the activity. Did the participants adhere to the execution of an activity to its specification ? They used three body sensors (Belt, Arm and Glove) to measure whether a certain activity was done right. Participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions:

Exactly according to the specification (Class A) Throwing the elbows to the front (Class B) *Lifting the dumbbell only halfway (Class C)* Lowering the dumbbell only halfway(Class D) *Throwing the hips to the front (Class E)

Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes. The exercises were performed by six male participants aged between 20-28 years, with little weight lifting experience.

For more information please read more here @http://groupware.les.inf.puc-rio.br/public/papers/2013.Velloso.QAR-WLE.pdf

Libraries

These are the list of R packages needed to run this code Assuming all installations are previously done

```
library(caret)
library(randomForest)
library(tidyverse)
library(mlbench)
library(janitor)
library(AppliedPredictiveModeling)
library(rattle)
library(rpart.plot)
library(knitr);
```

Data

```
load("/Users/Kawther/Desktop/DataScientist/Machine Learning/Final Project.RData")
## Set data URL
TrainURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
TestURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

## Download data
## and make sure it was not downloaded before
if (!file.exists("pml-training.csv")) {
  download.file(url = TrainURL, destfile = "pml-training.csv")
}
if (!file.exists("pml-testing.csv")) {
  download.file(url = TestURL, destfile = "pml-testing.csv")
}

## Read the  data
training <- read.csv("pml-training.csv", na.strings=c("NA",""), header=TRUE)
testing <- read.csv("pml-testing.csv",na.strings=c("NA",""), header=TRUE)

## Saving the Date the data got downloaded
dateDownloaded <- date()
```

The Data was downloaded on Sat Feb 04 18:19:16 2017

Exploratory Analysis

lets take a look of what the data looks like only in the training set

As per Prediction Study Design, the test set will not be used or even looked at. and will only be used one time to apply our predictor , however will use the same techniques to clean the data on both sets.

Clean Data

```
## get rid of all columns that have 50% NA values
cleantraining <- training[, -which(colMeans(is.na(training)) > 0.5)]
cleantesting <- testing[,-which(colMeans(is.na(testing)) > 0.5)]

## remove the first 7 columns/variable (not needed data)
cleantraining <- cleantraining[,8:length(colnames(cleantraining))]
cleantesting <- cleantesting[,8:length(colnames(cleantesting))]

## make sure both datasets have the same variables
CleanTrainingCol <- colnames(cleantraining)
CleanTestingCol <- colnames(cleantesting)
if (all.equal(CleanTrainingCol[1:length(CleanTrainingCol)-1],
CleanTestingCol[1:length(CleanTrainingCol)-1])) { print("The two datasets have the same columns/variabl
e") } else
  {
  print("WARNING: the two datasets dont have the same columns/variables")
}
```

```
## [1] "The two datasets have the same columns/variable"
```

```
DimensionCleanTraining <- dim(cleantraining)
DimensionCleanTesting <- dim (cleantesting)
```

The training data set has 19622 rows/obs and 53 columns/variables.

The testing data set has 20 rows/obs and 53 columns/variables.

There are a large number of variables that are not needed for our calculations. To reduce the number of variables, we need to only include the ones that have high correlation. Will use the NearZeroVar function to remove variables that dont have high correlations

```
nsv <- nearZeroVar(cleantraining, saveMetrics=TRUE)

## count how many True is nzv
CountTrueNZV <-length(which(nsv=="TRUE"))
```

There are 0 variables that have little variability and will likely not be good predictors. So that did not help much in eliminating any variable.

The Training data is still a large dataset, which gives us the opportunity to cross validate.

Having expiremented with Random Forrest to model this dataset previously, I found that it took more that 2 hours for my laptop to process. So based on StakOverFlow suggestion , I decided to divide my data into 4 datasets and then divide each new dataset into two , a training and a validation set. So will split the training set into two subsets: training and validation. Now will have three subsets: * The training sets will be used to fit the models * The validation sets will be used to estimate prediction error for model selection *The test set will be used for assessment of the generalization error of the final chosen model.

```
## Split the data in half
set.seed(01312017)
FirstSplit <- createDataPartition(y=cleantraining$classe, p=0.5, list=FALSE)
HalfDataA<- cleantraining[FirstSplit,]
HalfDataB <- cleantraining[-FirstSplit,]
dim(HalfDataA); dim(HalfDataB)
```

```
## [1] 9812   53
```

```
## [1] 9810   53
```

```
## Split the new data set into 2 again
set.seed(01312017)
SecondSplitA <- createDataPartition(y=HalfDataA$classe,p=0.5, list = FALSE)
Training1 <-HalfDataA[SecondSplitA,]
Training2 <- HalfDataA[-SecondSplitA,]
dim(Training1); dim(Training2)
```

```
## [1] 4907   53
```

```
## [1] 4905   53
```

```
set.seed(01312017)
SecondSplitB <- createDataPartition(y=HalfDataB$classe,p=0.5, list = FALSE)
Training3 <-HalfDataB[SecondSplitB,]
Training4 <- HalfDataB[-SecondSplitB,]
dim(Training3);dim(Training4)
```

```
## [1] 4906   53
```

```
## [1] 4904   53
## Now lets split the 4 new training datasets into training and validations set
## 60% Training 40% Validation

set.seed(01312017)
inTrain1 <- createDataPartition(y= Training1$classe, p=0.6,list=FALSE)
Training1 <- Training1[inTrain1,]
Validation1 <- Training1[-inTrain1,]

inTrain2 <- createDataPartition(y= Training2$classe, p=0.6,list=FALSE)
Training2 <- Training2[inTrain2,]
Validation2 <- Training2[-inTrain2,]

inTrain3 <- createDataPartition(y= Training3$classe, p=0.6,list=FALSE)
Training3 <- Training3[inTrain3,]
Validation3<- Training3[-inTrain3,]

inTrain4 <- createDataPartition(y= Training4$classe, p=0.6,list=FALSE)
Training4 <- Training4[inTrain4,]
Validation4 <- Training4[-inTrain4,]
```

Will try several modeling techiques:

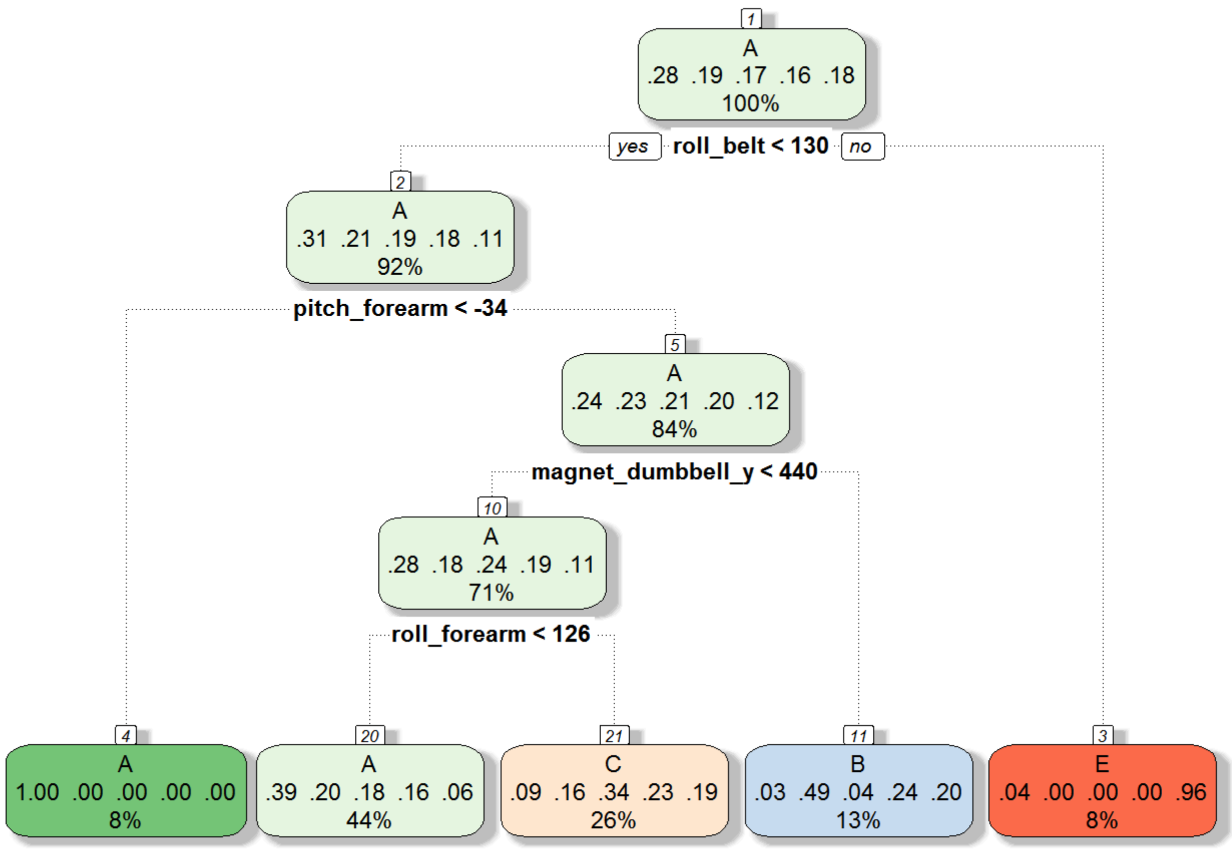
- **Recursive Partitioning and Regression Trees, using rpart()**

```
set.seed(01312017)

modFitA <- train(classe ~ ., data = Training1, method="rpart")
print(modFitA, digits=3)
```

```
## CART
##
## 2946 samples
##   52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 2946, 2946, 2946, 2946, 2946, 2946, ...
## Resampling results across tuning parameters:
##
##   cp      Accuracy  Kappa
##   0.0360  0.519     0.3846
##   0.0586  0.384     0.1563
##   0.1081  0.316     0.0478
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.036.
```

```
## Visualize the partition using fancyRpartPlot using rattle pkg.
library(rattle)
fancyRpartPlot(modFitA$finalModel)
```



Rattle 2017-Feb-04 18:19:36 Kat

- **Cross Validate with default bootstrapping**

```
## go with the default trainControl=boot (bootstrapping)
predictionsA<- predict(modFitA, newdata=Validation1)
CM<-confusionMatrix(predictionsA, Validation1$classe)
print(round(CM$overall,digits = 2))
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##           0.49           0.33           0.46           0.51           0.29
## AccuracyPValue  McnemarPValue
##           0.00            NaN
```

The Accuracy of this model; Recursive Partitioning and Regression Trees, using rpart() is 49% Not a good predictor.

Lets try to Standarize this dataset see if we get a better value Standardizing variable is to take the variables values and subtract their mean and then divide that whole quatity by the standard deviation (mean=0, sd=1)

```
set.seed(01312017)
modFitB <- train(classe ~ ., data = Training1, preProcess=c("center", "scale"), method="rpart")
print(modFitB, digits=3)
```

```
## CART
##
## 2946 samples
##   52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (52), scaled (52)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 2946, 2946, 2946, 2946, 2946, 2946, ...
## Resampling results across tuning parameters:
##
##   cp      Accuracy  Kappa
##   0.0360  0.519     0.3846
##   0.0586  0.384     0.1563
##   0.1081  0.316     0.0478
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.036.
```

```
predictionsB<- predict(modFitB, newdata=Validation1)
CMB<-confusionMatrix(predictionsB, Validation1$classe)
print(round(CMB$overall,digits = 2))
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##           0.49           0.33           0.46           0.51           0.29
## AccuracyPValue  McnemarPValue
##           0.00            NaN
```

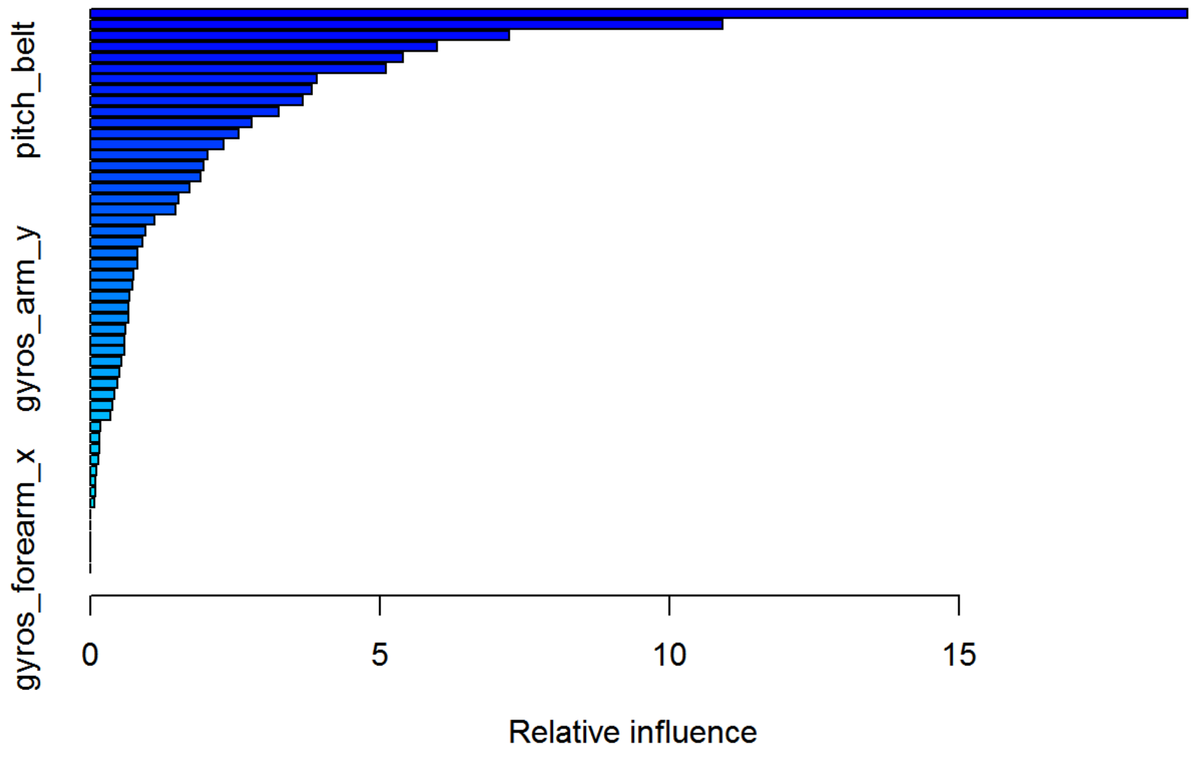
The Accuracy of the prediction with Standardizing process is 49% It did not improve much. Still a weak model.

- **Stochastic gradient boosting trees (gbm)**

```
set.seed(01312017)
ctrlgbm <- trainControl(method='cv', number=3, returnResamp='none', classProbs = TRUE)
## metrics use ROC instead of the default RMSE:
modFitgbm<- train(classe ~ ., data = Training1,
                  method="gbm",trControl=ctrlgbm,
                  metric = "ROC",
                  preProc = c("center", "scale"),
                  verbose=FALSE)
print(modFitgbm, digits=3)
```

```
## Stochastic Gradient Boosting
##
## 2946 samples
##   52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (52), scaled (52)
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 1964, 1964, 1964
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy  Kappa
##   1                   50      0.741     0.672
##   1                  100      0.803     0.751
##   1                  150      0.834     0.790
##   2                   50      0.827     0.781
##   2                  100      0.880     0.848
##   2                  150      0.902     0.876
##   3                   50      0.869     0.834
##   3                  100      0.912     0.888
##   3                  150      0.923     0.903
##
## 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.
```

```
summary(modFitgbm)
```


pitch_belt
gyros_arm_y
gyros_forearm_x
gyros_
0
5
10
15
Relative influence

```
##                                        var      rel.inf
## roll_belt                         roll_belt 18.95038537
## pitch_forearm                 pitch_forearm 10.92147882
## magnet_dumbbell_z         magnet_dumbbell_z  7.23780841
## roll_forearm                   roll_forearm  5.98481571
## yaw_belt                           yaw_belt  5.40364607
## magnet_dumbbell_y         magnet_dumbbell_y  5.10178070
## roll_dumbbell                 roll_dumbbell  3.91581122
## pitch_belt                       pitch_belt  3.83251709
## magnet_belt_z                 magnet_belt_z  3.67250653
## gyros_belt_z                   gyros_belt_z  3.25546949
## magnet_belt_y                 magnet_belt_y  2.77989463
## gyros_dumbbell_y           gyros_dumbbell_y  2.56722313
## accel_forearm_x             accel_forearm_x  2.30855089
## accel_dumbbell_x           accel_dumbbell_x  2.02830891
## magnet_dumbbell_x         magnet_dumbbell_x  1.95680924
## accel_dumbbell_y           accel_dumbbell_y  1.90701322
## yaw_arm                             yaw_arm  1.71300442
## accel_dumbbell_z           accel_dumbbell_z  1.52605254
## magnet_forearm_z           magnet_forearm_z  1.46981292
## magnet_belt_x                 magnet_belt_x  1.11171187
## roll_arm                           roll_arm  0.95511109
## magnet_arm_z                   magnet_arm_z  0.90298877
## magnet_arm_x                   magnet_arm_x  0.81231667
## magnet_arm_y                   magnet_arm_y  0.81011219
## magnet_forearm_x           magnet_forearm_x  0.74781718
## accel_belt_z                   accel_belt_z  0.73517971
## magnet_forearm_y           magnet_forearm_y  0.68154990
## gyros_arm_x                     gyros_arm_x  0.66071888
## gyros_arm_y                     gyros_arm_y  0.65189990
## pitch_dumbbell               pitch_dumbbell  0.60146274
## total_accel_dumbbell total_accel_dumbbell  0.58695838
## total_accel_belt           total_accel_belt  0.58282515
## accel_arm_x                     accel_arm_x  0.54098393
## accel_forearm_z             accel_forearm_z  0.50753208
## accel_arm_z                     accel_arm_z  0.46941083
## total_accel_forearm     total_accel_forearm  0.42236771
## accel_forearm_y             accel_forearm_y  0.38874085
## gyros_dumbbell_x           gyros_dumbbell_x  0.34261705
## gyros_dumbbell_z           gyros_dumbbell_z  0.16691377
## gyros_forearm_y             gyros_forearm_y  0.15230568
## gyros_belt_x                   gyros_belt_x  0.15000873
## gyros_belt_y                   gyros_belt_y  0.13361722
## pitch_arm                         pitch_arm  0.10373201
## yaw_dumbbell                   yaw_dumbbell  0.08597210
## gyros_forearm_z             gyros_forearm_z  0.08269463
## gyros_arm_z                    gyros_arm_z  0.07956170
## accel_belt_x                 accel_belt_x  0.00000000
## accel_belt_y                 accel_belt_y  0.00000000
## total_accel_arm           total_accel_arm  0.00000000
## accel_arm_y                   accel_arm_y  0.00000000
## yaw_forearm                   yaw_forearm  0.00000000
## gyros_forearm_x           gyros_forearm_x  0.00000000
```

```
predictionsgbm<- predict(modFitgbm, newdata=Validation1)
CMgbm<-confusionMatrix(predictionsgbm, Validation1$classe)
print(round(CMgbm$overall,digits = 2))
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##           0.98           0.98           0.97           0.99           0.29
## AccuracyPValue  McnemarPValue
##           0.00            NaN
```

The Accuracy of Stochastic gradient boosting trees (gbm) predictor is 98% Very good predictor, but lets try against another validation set, incase we overfit the last one

```
predictionsgbm2<- predict(modFitgbm, newdata=Validation2)
CMgbm2<-confusionMatrix(predictionsgbm2, Validation2$classe)
print(round(CMgbm2$overall,digits = 2))
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##           0.94           0.93           0.93           0.95           0.28
## AccuracyPValue  McnemarPValue
##           0.00           0.01
```

The Accuracy of Stochastic gradient boosting trees (gbm) predictor is 94% Still a high Accuracy and seems to be a good model.

Random Forest Model

Random Forest is one of the most used and highly accurate model highly used by Kaggle competitors. So will give it a try. Will use a 5-fold cross validation (K-fold cross validation) so that we dont overfit our model.

```
set.seed(01312017)
## Use a trainControl to have control over the search grid.
## 5-fold cross validation (K-fold cross validation)
ctrl <- trainControl("cv", number = 5, verboseIter = FALSE)
modFitC<- train(classe ~ ., method="rf", trControl=ctrl, data=Training2)
print(modFitC, digits=3)
```

```
## Random Forest
##
## 2945 samples
##   52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2355, 2357, 2356, 2355, 2357
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa
##    2    0.948     0.934
##   27    0.952     0.939
##   52    0.941     0.926
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

The accuracy of Random Forest model using 5-fold cross validation is 95% using mtry =27. Much better Accuracy than the Recursive Partitioning and Regression Trees.

Lets try it again with another set validation

```
head(getTree(modFitC$finalModel, k=27))
```

```
##   left daughter right daughter split var split point status prediction
## 1             2              3         1      130.50      1          0
## 2             4              5        40      124.50      1          0
## 3             6              7        15       58.40      1          0
## 4             8              9        41       -0.25      1          0
## 5            10             11        38      280.50      1          0
## 6             0              0         0        0.00     -1          5
```

```
predictionsC <- predict(modFitC, newdata=Validation3)
CMrf <-confusionMatrix(predictionsC, Validation3$classe)
print(CMrf, digits=4)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 338   4   0   1   0
##          B   3 203   7   1   3
##          C   1  11 203   3   2
##          D   0   1   5 183   2
##          E   0   0   0   2 210
##
## Overall Statistics
##
##                Accuracy : 0.9611
##                  95% CI : (0.9485, 0.9714)
##     No Information Rate : 0.2891
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9508
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9883   0.9269   0.9442   0.9632   0.9677
## Specificity            0.9941   0.9855   0.9824   0.9919   0.9979
## Pos Pred Value         0.9854   0.9355   0.9227   0.9581   0.9906
## Neg Pred Value         0.9952   0.9834   0.9875   0.9929   0.9928
## Prevalence             0.2891   0.1851   0.1817   0.1606   0.1834
## Detection Rate         0.2857   0.1716   0.1716   0.1547   0.1775
## Detection Prevalence   0.2899   0.1834   0.1860   0.1615   0.1792
## Balanced Accuracy      0.9912   0.9562   0.9633   0.9776   0.9828
```

Out of sample error: The error rate you get on a new dataset (Validation) The accuracy of Random Forest model using 5-fold cross validation is 96% The Out of Sample error rate is 3.8884193 % That’s pretty much around what it was expected.

Summary

it looks like both Boosting with Tress (gbm) and Random Forest(rf) did better that the Recursive Partitioning and Regression Trees, using rpart(). The out of sample error rate is 0.0388842 The data was very large and I had to split it in 4 subsets so that my PC can handle the calculations.

```
AccuracyResults <- data.frame(
  Model = c('rpart', 'gbm', 'rf'),
  Accuracy = rbind(CM$overall[1], CMgbm$overall[1], CMrf$overall[1])
)
print(AccuracyResults)
```

```
##   Model  Accuracy
## 1 rpart 0.4856902
## 2   gbm 0.9806397
## 3    rf 0.9611158
```