Human Activity Recognition - Course Project

Sridhar Pilli

March 24, 2016

Summary

This project describes the analysis of human activity regonition data to predict the quality of a particular type of exercise. In this report various aspects of feature selection, building and evaluation of models would be covered and a final model would be selected.

Data Analysis

First lets read the data and carry out exploratory analysis to select appropriate level1 and level2 covariates. At the end split the data into training and crossvalidation set.

Structure of the Data

The publication of the authors descibes data containing **actual measurements** from the sensors placed on four different locations and **derived quanitities** from these actual measurements. Specifically there are 1. **4** locations where sensors are placed - Forearm, Arm, Belt, Dumbell 2. **12** Actual measurements taken from each location - Euler's angles (Roll, Pitch, Yaw), Acceleration(X,Y,Z), Gyroscope(X,Y,Z), Magnetometer(X,Y,Z) 3. **8** Derived quanitites for each of the Euler's angles - Mean, Variance, Standard Deviation, Min, Max, Amplitude, Kurtosis, Skewness 4. **2** Derived quantities for acceleration overall - Total and Variance. 5. **7** Housekeeping variables such as timestamp, username etc. Of this the relevant one would be the "new_window".

Reading and cleaning the data

Read the data from csv file

```
setwd("~/Documents/Git/Github/Project8/")
# Read the csv files. Notice the data has quite a few empty cells. These are converted to NA while read
trainHAR <- read.csv("pml-training.csv", na.strings = "", stringsAsFactors = FALSE)
testHAR <- read.csv("pml-testing.csv", na.strings = "", stringsAsFactors = FALSE)
```

Level1 : Rawdata -> Covariates

Convert the missing values to NAs, Divide-by-zeros to Inf and remove the first 7 columns as there isn't meaningful information there. Ideally had the testing data contained entries for the derived quantities such as mean, variance of measurementes we could have kept the num_window and new_window columns to group the data into windows and then compute the averages.

```
for(var in names(trainHAR)) { trainHAR[which(trainHAR[,var]=="#DIV/0!"),var] <- Inf }
for(i in 8:ncol(trainHAR)-1) { trainHAR[,i] <- as.numeric(trainHAR[,i]) }
trainHAR$classe <- as.factor(trainHAR$classe)
train1 <- trainHAR[8:160]
```

```r
test1 <- testHAR[,8:159]
for(i in 1:152) { test1[,i] <- as.numeric(test1[,i]) }
```

Lets get rid of those columns where there is either NA or Inf.

```r
id1 <- sapply(train1[,1:152],function(x) { (mean(x)==Inf)|(is.na(mean(x))) })
id2 <- sapply(test1[,1:152],function(x) { (mean(x)==Inf)|(is.na(mean(x))) })
n1 <- names(train1); n2 <- names(test1)
n <- union(n1[id1],n2[id2]) # these are the columns to get rid
train1 <- train1[,setdiff(n1,n)]
dim(train1)
```

```
## [1] 19622    53
```

```r
names(train1)
```

```
##  [1] "roll_belt"            "pitch_belt"           "yaw_belt"
##  [4] "total_accel_belt"     "gyros_belt_x"         "gyros_belt_y"
##  [7] "gyros_belt_z"         "accel_belt_x"         "accel_belt_y"
## [10] "accel_belt_z"         "magnet_belt_x"        "magnet_belt_y"
## [13] "magnet_belt_z"        "roll_arm"             "pitch_arm"
## [16] "yaw_arm"              "total_accel_arm"      "gyros_arm_x"
## [19] "gyros_arm_y"          "gyros_arm_z"          "accel_arm_x"
## [22] "accel_arm_y"          "accel_arm_z"          "magnet_arm_x"
## [25] "magnet_arm_y"         "magnet_arm_z"         "roll_dumbbell"
## [28] "pitch_dumbbell"       "yaw_dumbbell"         "total_accel_dumbbell"
## [31] "gyros_dumbbell_x"     "gyros_dumbbell_y"     "gyros_dumbbell_z"
## [34] "accel_dumbbell_x"     "accel_dumbbell_y"     "accel_dumbbell_z"
## [37] "magnet_dumbbell_x"    "magnet_dumbbell_y"    "magnet_dumbbell_z"
## [40] "roll_forearm"         "pitch_forearm"        "yaw_forearm"
## [43] "total_accel_forearm"  "gyros_forearm_x"      "gyros_forearm_y"
## [46] "gyros_forearm_z"      "accel_forearm_x"      "accel_forearm_y"
## [49] "accel_forearm_z"      "magnet_forearm_x"     "magnet_forearm_y"
## [52] "magnet_forearm_z"     "classe"
```

```r
test1 <- test1[,setdiff(n2,n)]
dim(test1)
```

```
## [1] 20 52
```

```r
names(test1)
```

```
##  [1] "roll_belt"            "pitch_belt"           "yaw_belt"
##  [4] "total_accel_belt"     "gyros_belt_x"         "gyros_belt_y"
##  [7] "gyros_belt_z"         "accel_belt_x"         "accel_belt_y"
## [10] "accel_belt_z"         "magnet_belt_x"        "magnet_belt_y"
## [13] "magnet_belt_z"        "roll_arm"             "pitch_arm"
## [16] "yaw_arm"              "total_accel_arm"      "gyros_arm_x"
## [19] "gyros_arm_y"          "gyros_arm_z"          "accel_arm_x"
## [22] "accel_arm_y"          "accel_arm_z"          "magnet_arm_x"
```

```
## [25] "magnet_arm_y"         "magnet_arm_z"         "roll_dumbbell"
## [28] "pitch_dumbbell"       "yaw_dumbbell"         "total_accel_dumbbell"
## [31] "gyros_dumbbell_x"     "gyros_dumbbell_y"     "gyros_dumbbell_z"
## [34] "accel_dumbbell_x"     "accel_dumbbell_y"     "accel_dumbbell_z"
## [37] "magnet_dumbbell_x"    "magnet_dumbbell_y"    "magnet_dumbbell_z"
## [40] "roll_forearm"         "pitch_forearm"        "yaw_forearm"
## [43] "total_accel_forearm"  "gyros_forearm_x"      "gyros_forearm_y"
## [46] "gyros_forearm_z"      "accel_forearm_x"      "accel_forearm_y"
## [49] "accel_forearm_z"      "magnet_forearm_x"     "magnet_forearm_y"
## [52] "magnet_forearm_z"
```

At the end of this step we are down to 53 covariates from 160 in the training set

Level2 : Tidy covariates -> New covariates

From the tidy covariates set let first figure out which ones have zero variance and can be removed right away. Note we are monitoring the stand alone variance without comparing other covariates. Lets keep those in with large variance.

```r
library(caret)
```

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

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

```r
nzv <- nearZeroVar(train1,saveMetrics = TRUE)
nzv
```

```
##                      freqRatio percentUnique zeroVar   nzv
## roll_belt             1.101904     6.7781062   FALSE FALSE
## pitch_belt            1.036082     9.3772296   FALSE FALSE
## yaw_belt              1.058480     9.9734991   FALSE FALSE
## total_accel_belt      1.063160     0.1477933   FALSE FALSE
## gyros_belt_x          1.058651     0.7134849   FALSE FALSE
## gyros_belt_y          1.144000     0.3516461   FALSE FALSE
## gyros_belt_z          1.066214     0.8612782   FALSE FALSE
## accel_belt_x          1.055412     0.8357966   FALSE FALSE
## accel_belt_y          1.113725     0.7287738   FALSE FALSE
## accel_belt_z          1.078767     1.5237998   FALSE FALSE
## magnet_belt_x         1.090141     1.6664968   FALSE FALSE
## magnet_belt_y         1.099688     1.5187035   FALSE FALSE
## magnet_belt_z         1.006369     2.3290184   FALSE FALSE
## roll_arm             52.338462    13.5256345   FALSE FALSE
## pitch_arm            87.256410    15.7323412   FALSE FALSE
## yaw_arm              33.029126    14.6570176   FALSE FALSE
## total_accel_arm       1.024526     0.3363572   FALSE FALSE
## gyros_arm_x           1.015504     3.2769341   FALSE FALSE
## gyros_arm_y           1.454369     1.9162165   FALSE FALSE
## gyros_arm_z           1.110687     1.2638875   FALSE FALSE
## accel_arm_x           1.017341     3.9598410   FALSE FALSE
## accel_arm_y           1.140187     2.7367241   FALSE FALSE
```

```
## accel_arm_z           1.128000     4.0362858   FALSE FALSE
## magnet_arm_x          1.000000     6.8239731   FALSE FALSE
## magnet_arm_y          1.056818     4.4439914   FALSE FALSE
## magnet_arm_z          1.036364     6.4468454   FALSE FALSE
## roll_dumbbell         1.022388    84.2065029   FALSE FALSE
## pitch_dumbbell        2.277372    81.7449801   FALSE FALSE
## yaw_dumbbell          1.132231    83.4828254   FALSE FALSE
## total_accel_dumbbell  1.072634     0.2191418   FALSE FALSE
## gyros_dumbbell_x      1.003268     1.2282132   FALSE FALSE
## gyros_dumbbell_y      1.264957     1.4167771   FALSE FALSE
## gyros_dumbbell_z      1.060100     1.0498420   FALSE FALSE
## accel_dumbbell_x      1.018018     2.1659362   FALSE FALSE
## accel_dumbbell_y      1.053061     2.3748853   FALSE FALSE
## accel_dumbbell_z      1.133333     2.0894914   FALSE FALSE
## magnet_dumbbell_x     1.098266     5.7486495   FALSE FALSE
## magnet_dumbbell_y     1.197740     4.3012945   FALSE FALSE
## magnet_dumbbell_z     1.020833     3.4451126   FALSE FALSE
## roll_forearm         11.589286    11.0895933   FALSE FALSE
## pitch_forearm        65.983051    14.8557741   FALSE FALSE
## yaw_forearm          15.322835    10.1467740   FALSE FALSE
## total_accel_forearm   1.128928     0.3567424   FALSE FALSE
## gyros_forearm_x       1.059273     1.5187035   FALSE FALSE
## gyros_forearm_y       1.036554     3.7763735   FALSE FALSE
## gyros_forearm_z       1.122917     1.5645704   FALSE FALSE
## accel_forearm_x       1.126437     4.0464784   FALSE FALSE
## accel_forearm_y       1.059406     5.1116094   FALSE FALSE
## accel_forearm_z       1.006250     2.9558659   FALSE FALSE
## magnet_forearm_x      1.012346     7.7667924   FALSE FALSE
## magnet_forearm_y      1.246914     9.5403119   FALSE FALSE
## magnet_forearm_z      1.000000     8.5771073   FALSE FALSE
## classe                1.469581     0.0254816   FALSE FALSE
```

From the above output the "nzv" value for all the covaraites is FALSE indicating that their is no variable whose variance is small enough to be discarded.

Next lets use findCorrelation() function in Caret package to figure out which variables can be dropped. The idea here is the measure the correlation of a particular covariate with other covariates and then discard those variates whose correlation is high. For this I am going to use a cutoff of 0.9. In other words if variable x1 and x2 are highly correlated with absolute correlation >0.9 then I am going to drop one of these variables.

```r
library(caret)
cor_vals <- cor(train1[,1:52])
drop_vars <- findCorrelation(cor_vals,cutoff=0.9)
colNames <- c(names(train1)[setdiff(1:52,drop_vars)], "classe")
train2 <- train1[,colNames]
colNames
```

```
##  [1] "pitch_belt"           "yaw_belt"             "total_accel_belt"
##  [4] "gyros_belt_x"         "gyros_belt_y"         "gyros_belt_z"
##  [7] "magnet_belt_x"        "magnet_belt_y"        "magnet_belt_z"
## [10] "roll_arm"             "pitch_arm"            "yaw_arm"
## [13] "total_accel_arm"      "gyros_arm_y"          "gyros_arm_z"
## [16] "accel_arm_x"          "accel_arm_y"          "accel_arm_z"
## [19] "magnet_arm_x"         "magnet_arm_y"         "magnet_arm_z"
```

```
## [22] "roll_dumbbell"        "pitch_dumbbell"       "yaw_dumbbell"
## [25] "total_accel_dumbbell" "gyros_dumbbell_y"     "accel_dumbbell_x"
## [28] "accel_dumbbell_y"     "accel_dumbbell_z"     "magnet_dumbbell_x"
## [31] "magnet_dumbbell_y"    "magnet_dumbbell_z"    "roll_forearm"
## [34] "pitch_forearm"        "yaw_forearm"          "total_accel_forearm"
## [37] "gyros_forearm_x"      "gyros_forearm_y"      "gyros_forearm_z"
## [40] "accel_forearm_x"      "accel_forearm_y"      "accel_forearm_z"
## [43] "magnet_forearm_x"     "magnet_forearm_y"     "magnet_forearm_z"
## [46] "classe"
```

At the end of this step we are down to 46 covariates from 160 at the start.

Split the data into training and cross validation set

Lets split the cleaned up training data into 75% training and 25% cross validation datasets.

```
set.seed(1234)
inTrain <- createDataPartition(train2$classe,p=0.75,list=FALSE)
training <- train2[inTrain,]
validation <- train2[-inTrain,]
dim(training)
```

```
## [1] 14718    46
```

```
dim(validation)
```

```
## [1] 4904   46
```

Model fitting

The approach I would like to take here is to evaluate a bunch of classifiers and an ensemble of them to see which one gives higher accuracy and lower error rates. ## CART

```
require(caret);
set.seed(34523)
fit1 <- train(classe~.,method="rpart",data=training)
```

```
## Loading required package: rpart
```

```
fit1
```

```
## CART
##
## 14718 samples
##    45 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
```

```
## Summary of sample sizes: 14718, 14718, 14718, 14718, 14718, 14718, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa      Accuracy SD  Kappa SD
##   0.02307035  0.5667032  0.4505993  0.03696502   0.05085124
##   0.02610842  0.5471537  0.4258219  0.03687745   0.04955390
##   0.04196335  0.4024594  0.2013193  0.11048712   0.18509401
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.02307035.
```

```
confusionMatrix(validation$classe, predict(fit1,validation))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 858  17 375 143   2
##          B 154 409 332  51   3
##          C  18  20 811   6   0
##          D  49  59 345 286  65
##          E  11 153 361  71 305
##
## Overall Statistics
##
##                Accuracy : 0.5442
##                  95% CI : (0.5302, 0.5583)
##     No Information Rate : 0.4535
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4296
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7872   0.6216   0.3647  0.51346  0.81333
## Specificity            0.8592   0.8728   0.9836  0.88084  0.86840
## Pos Pred Value         0.6151   0.4310   0.9485  0.35572  0.33851
## Neg Pred Value         0.9339   0.9370   0.6510  0.93390  0.98251
## Prevalence             0.2223   0.1342   0.4535  0.11358  0.07647
## Detection Rate         0.1750   0.0834   0.1654  0.05832  0.06219
## Detection Prevalence   0.2845   0.1935   0.1743  0.16395  0.18373
## Balanced Accuracy      0.8232   0.7472   0.6741  0.69715  0.84087
```

The accuracy is about **50%** and the error is **50%** (100%-accuracy). This is not a great number to start with. Lets see if this gets any better if we centre and scale the data.

```
fit2 <- train(classe~., method="rpart", preProcess=c("center","scale"), data=training)
fit2
```

```
## CART
##
```

```
## 14718 samples
##    45 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (45), scaled (45)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 14718, 14718, 14718, 14718, 14718, 14718, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa      Accuracy SD  Kappa SD
##   0.02307035  0.5449790  0.4186585  0.03071313   0.04773702
##   0.02610842  0.5280106  0.3942526  0.02689763   0.04185966
##   0.04196335  0.4483918  0.2759832  0.07422485   0.12701728
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.02307035.
```

```
confusionMatrix(validation$classe, predict(fit2,validation))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 858  17 375 143   2
##          B 154 409 332  51   3
##          C  18  20 811   6   0
##          D  49  59 345 286  65
##          E  11 153 361  71 305
##
## Overall Statistics
##
##                Accuracy : 0.5442
##                  95% CI : (0.5302, 0.5583)
##     No Information Rate : 0.4535
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4296
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7872   0.6216   0.3647  0.51346  0.81333
## Specificity            0.8592   0.8728   0.9836  0.88084  0.86840
## Pos Pred Value         0.6151   0.4310   0.9485  0.35572  0.33851
## Neg Pred Value         0.9339   0.9370   0.6510  0.93390  0.98251
## Prevalence             0.2223   0.1342   0.4535  0.11358  0.07647
## Detection Rate         0.1750   0.0834   0.1654  0.05832  0.06219
## Detection Prevalence   0.2845   0.1935   0.1743  0.16395  0.18373
## Balanced Accuracy      0.8232   0.7472   0.6741  0.69715  0.84087
```

This model is no different than the previous one.

Random Forest

Since the model building is taking quite a bit of time on my computer I am going to train on a much smaller set and validate on a slightly larger set. I am going to repeat this experiment a few times to randomize the test.

```r
acc_val <- numeric(5)
set.seed(12121)
for (i in 1:5) {
    # Subsample training data and For each class type pick about 100 rows of data
    train_rf <- training[sample(which(train2$classe=="A"),100),]
    train_rf <- rbind(train_rf, training[sample(which(train2$classe=="B"),100),])
    train_rf <- rbind(train_rf, training[sample(which(train2$classe=="C"),100),])
    train_rf <- rbind(train_rf, training[sample(which(train2$classe=="D"),100),])
    train_rf <- rbind(train_rf, training[sample(which(train2$classe=="E"),100),])
    # Fit a model
    fit3 <- train(classe~.,method="rf",data=train_rf)
    # Predict using the entire validation data and measure the accuracy.
    acc_val[i] <- mean(validation$classe==predict(fit3,validation))
}
```

```
## Loading required package: randomForest

## randomForest 4.6-12

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

##
## Attaching package: 'randomForest'

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

```r
acc_val
```

```
## [1] 0.7491843 0.7661093 0.7858891 0.7989396 0.7689641
```

```r
fit3
```

```
## Random Forest
##
## 500 samples
##  45 predictor
##   5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 351, 351, 351, 351, 351, 351, ...
## Resampling results across tuning parameters:
##
```

```
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD  
##    2    0.7308588  0.6632262  0.03404927   0.04313492
##   23    0.7398934  0.6742756  0.03919438   0.04922428
##   45    0.7331535  0.6658962  0.03959702   0.04945680
## 
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 23.
```

```r
confusionMatrix(validation$classe, predict(fit3, validation))
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1093   94   69  130    9
##          B   56  664  124   58   47
##          C   31   39  716   58   11
##          D   20   41  118  595   30
##          E   17   56   77   49  702
## 
## Overall Statistics
##                                           
##                Accuracy : 0.7688          
##                  95% CI : (0.7567, 0.7805)
##     No Information Rate : 0.2482          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.7092          
##  Mcnemar's Test P-Value : < 2.2e-16       
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8981   0.7427   0.6486   0.6685   0.8786
## Specificity            0.9181   0.9289   0.9634   0.9479   0.9515
## Pos Pred Value         0.7835   0.6997   0.8374   0.7400   0.7791
## Neg Pred Value         0.9647   0.9418   0.9042   0.9280   0.9758
## Prevalence             0.2482   0.1823   0.2251   0.1815   0.1629
## Detection Rate         0.2229   0.1354   0.1460   0.1213   0.1431
## Detection Prevalence   0.2845   0.1935   0.1743   0.1639   0.1837
## Balanced Accuracy      0.9081   0.8358   0.8060   0.8082   0.9151
```

This is a better model than the CART version. Accuracy is about **80%** and error rate is about **20%**. Since accuracy of the model isn't signifncantly different from run to run, lets select the model from last iteration as a candaidate.

Boosting with trees

```r
acc_val <- numeric(5)
set.seed(23232)
for (i in 1:5) {
```

```r
    # Subsample training data and For each class type pick about 100 rows of data
    train_rf <- training[sample(which(train2$classe=="A"),100),]
    train_rf <- rbind(train_rf, training[sample(which(train2$classe=="B"),100),])
    train_rf <- rbind(train_rf, training[sample(which(train2$classe=="C"),100),])
    train_rf <- rbind(train_rf, training[sample(which(train2$classe=="D"),100),])
    train_rf <- rbind(train_rf, training[sample(which(train2$classe=="E"),100),])
    # Fit a model
    fit4 <- train(classe~.,method="gbm",data=train_rf, verbose=FALSE)
    # Predict using the entire validation data and measure the accuracy.
    acc_val[i] <- mean(validation$classe==predict(fit4,validation))
}
```

```
## Loading required package: gbm
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.1
```

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

```r
acc_val
```

```
## [1] 0.8019984 0.7730424 0.7412316 0.7842577 0.7714111
```

```r
fit4
```

```
## Stochastic Gradient Boosting
##
## 500 samples
##  45 predictor
##   5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 355, 355, 355, 355, 355, 355, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa      Accuracy SD
##   1                   50      0.6178830  0.5216989  0.04218385
```

```
##   1                  100      0.6532226  0.5659691  0.03383247
##   1                  150      0.6690519  0.5856809  0.03675320
##   2                   50      0.6628088  0.5776752  0.03609515
##   2                  100      0.6894592  0.6111148  0.03807891
##   2                  150      0.7019732  0.6265265  0.03624132
##   3                   50      0.6853063  0.6055035  0.03094699
##   3                  100      0.7078983  0.6336624  0.03611259
##   3                  150      0.7229936  0.6526640  0.03578846
##   Kappa SD
##   0.05209314
##   0.04172170
##   0.04488934
##   0.04511410
##   0.04728891
##   0.04489273
##   0.03892326
##   0.04506187
##   0.04461362
##
## 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.
```

```r
confusionMatrix(validation$classe, predict(fit4,validation))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1138   50   80   58   69
##          B  112  644   64   38   91
##          C   37   64  651   58   45
##          D   29   24   77  631   43
##          E   18   82   57   25  719
##
## Overall Statistics
##
##                Accuracy : 0.7714
##                  95% CI : (0.7594, 0.7831)
##     No Information Rate : 0.272
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7114
##  Mcnemar's Test P-Value : 2.608e-15
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8531   0.7454   0.7008   0.7790   0.7435
## Specificity            0.9280   0.9245   0.9487   0.9577   0.9538
## Pos Pred Value         0.8158   0.6786   0.7614   0.7848   0.7980
```

```
## Neg Pred Value         0.9441   0.9444   0.9313   0.9563   0.9380
## Prevalence             0.2720   0.1762   0.1894   0.1652   0.1972
## Detection Rate         0.2321   0.1313   0.1327   0.1287   0.1466
## Detection Prevalence   0.2845   0.1935   0.1743   0.1639   0.1837
## Balanced Accuracy      0.8905   0.8349   0.8247   0.8684   0.8487
```

This is as good as the Random Forest model and better than CART. Accuracy is about **80%** and error rate is about **20%**. Since accuracy of the model isn't signifncantly different from run to run, lets select the model from last iteration as a candidate.

Combining all three models

Lets combine all three models above and see if we get any better performance.

```
df <- data.frame(rf    = predict(fit3,validation),
                 gbm   = predict(fit4,validation),
                 classe = validation$classe)
set.seed(45232)
fitAll <- train(classe~., method="rf", data=df[sample(1:4904,200),],verbose=FALSE)
fitAll
```

```
## Random Forest
##
## 200 samples
##   2 predictor
##   5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 200, 200, 200, 200, 200, 200, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##   2     0.8116918  0.7602542  0.04794334   0.06130849
##   5     0.7949304  0.7388956  0.04441801   0.05714319
##   8     0.7893029  0.7317879  0.04781198   0.06125750
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

Again the overall model is as good as either Random forest or the Boosted trees at accuracy of about 80%.

```
#save all the models.
saveRDS(fit3, file = "RandomForest_model.rds")
saveRDS(fit4, file = "BoostingTrees_model.rds")
saveRDS(fitAll, file = "FinalModel.rds" )
```

Predict outcome on the test dataset

Pick up relevant covariates needed for the model and then run through the model.

```r
test2 <- test1[,colNames[1:45]]
testDF <- data.frame(rf   = predict(fit3,test2),
                     gbm  = predict(fit4,test2))
predict(fitAll,testDF)
```

```
##  [1] C C B A A C D D A A C C B A E E A B A B
## Levels: A B C D E
```

Reference

The data for this project has been generously provided from the source http://groupware.les.inf.puc-rio.br/har

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.

Read more: http://groupware.les.inf.puc-rio.br/har#ixzz43rRuixaU