Practical_Machine_Learning_Human_Activity_Project

Venkat

8/11/2019

Overview

In this project, our goal is 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 (see the section on the Weight Lifting Exercise Dataset).

Data Processing

The data for this project come from this source: http://groupware.les.inf.puc-rio.br/har

###Load the data and alalyze

```
train_file="./data/pml-training.csv"
test_file="./data/pml-testing.csv"
train_url <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
test_url  <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

#check if file exists, else download it
if(!file.exists(train_file))
{
  download.file(train_url,train_file,method="auto")
}
if(!file.exists(test_file))
{
  download.file(test_url,test_file,method="auto")
}
##Load the data and Analyze
#train_data <- read.csv(train_file)
#head(train_data)
#dim(train_data)
#colnames(train_data)

## The data contains spaces, NA and DIV/0 values. make them na
train_data <- read.csv(train_file, na.strings=c("", " ","#DIV/0!","NA"))
test_data <- read.csv(test_file, na.strings=c("", " ","#DIV/0!","NA"))


dim(train_data)
```

```
## [1] 19622   160
```

```
#sapply(train_data, class)
str(head(train_data,10))
```

```
## 'data.frame':    10 obs. of  160 variables:
##  $ X                       : int  1 2 3 4 5 6 7 8 9 10
##  $ user_name               : Factor w/ 6 levels "adelmo","carlitos",..: 2 2 2 2 2 2 2 2 2 2
##  $ raw_timestamp_part_1    : int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 1
```

```
##  $ raw_timestamp_part_2    : int  788290 808298 820366 120339 196328 304277 368296 440390 484323 4844
##  $ cvtd_timestamp          : Factor w/ 20 levels "02/12/2011 13:32",..: 9 9 9 9 9 9 9 9 9 9
##  $ new_window              : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1
##  $ num_window              : int  11 11 11 12 12 12 12 12 12 12
##  $ roll_belt               : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45
##  $ pitch_belt              : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17
##  $ yaw_belt                : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4
##  $ total_accel_belt        : int  3 3 3 3 3 3 3 3 3 3
##  $ kurtosis_roll_belt      : num  NA NA NA NA NA NA NA NA NA NA
##  $ kurtosis_picth_belt     : num  NA NA NA NA NA NA NA NA NA NA
##  $ kurtosis_yaw_belt       : logi  NA NA NA NA NA NA ...
##  $ skewness_roll_belt      : num  NA NA NA NA NA NA NA NA NA NA
##  $ skewness_roll_belt.1    : num  NA NA NA NA NA NA NA NA NA NA
##  $ skewness_yaw_belt       : logi  NA NA NA NA NA NA ...
##  $ max_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA
##  $ max_picth_belt          : int  NA NA NA NA NA NA NA NA NA NA
##  $ max_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA
##  $ min_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA
##  $ min_pitch_belt          : int  NA NA NA NA NA NA NA NA NA NA
##  $ min_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA
##  $ amplitude_roll_belt     : num  NA NA NA NA NA NA NA NA NA NA
##  $ amplitude_pitch_belt    : int  NA NA NA NA NA NA NA NA NA NA
##  $ amplitude_yaw_belt      : num  NA NA NA NA NA NA NA NA NA NA
##  $ var_total_accel_belt    : num  NA NA NA NA NA NA NA NA NA NA
##  $ avg_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA
##  $ stddev_roll_belt        : num  NA NA NA NA NA NA NA NA NA NA
##  $ var_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA
##  $ avg_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA
##  $ stddev_pitch_belt       : num  NA NA NA NA NA NA NA NA NA NA
##  $ var_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA
##  $ avg_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA
##  $ stddev_yaw_belt         : num  NA NA NA NA NA NA NA NA NA NA
##  $ var_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA
##  $ gyros_belt_x            : num  0 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 0.03
##  $ gyros_belt_y            : num  0 0 0 0 0.02 0 0 0 0 0
##  $ gyros_belt_z            : num  -0.02 -0.02 -0.02 -0.03 -0.02 -0.02 -0.02 -0.02 -0.02 0
##  $ accel_belt_x            : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21
##  $ accel_belt_y            : int  4 4 5 3 2 4 3 4 2 4
##  $ accel_belt_z            : int  22 22 23 21 24 21 21 21 24 22
##  $ magnet_belt_x           : int  -3 -7 -2 -6 -6 0 -4 -2 1 -3
##  $ magnet_belt_y           : int  599 608 600 604 600 603 599 603 602 609
##  $ magnet_belt_z           : int  -313 -311 -305 -310 -302 -312 -311 -313 -312 -308
##  $ roll_arm                : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128
##  $ pitch_arm               : num  22.5 22.5 22.5 22.1 22.1 22 21.9 21.8 21.7 21.6
##  $ yaw_arm                 : num  -161 -161 -161 -161 -161 -161 -161 -161 -161 -161
##  $ total_accel_arm         : int  34 34 34 34 34 34 34 34 34 34
##  $ var_accel_arm           : num  NA NA NA NA NA NA NA NA NA NA
##  $ avg_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA
##  $ stddev_roll_arm         : num  NA NA NA NA NA NA NA NA NA NA
##  $ var_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA
##  $ avg_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA
##  $ stddev_pitch_arm        : num  NA NA NA NA NA NA NA NA NA NA
##  $ var_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA
##  $ avg_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA
```

```
##  $ stddev_yaw_arm          : num  NA NA NA NA NA NA NA NA NA NA
##  $ var_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA
##  $ gyros_arm_x             : num  0 0.02 0.02 0.02 0 0.02 0 0.02 0.02 0.02
##  $ gyros_arm_y             : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03
##  $ gyros_arm_z             : num  -0.02 -0.02 -0.02 0.02 0 0 0 0 -0.02 -0.02
##  $ accel_arm_x             : int  -288 -290 -289 -289 -289 -289 -289 -289 -288 -288
##  $ accel_arm_y             : int  109 110 110 111 111 111 111 111 109 110
##  $ accel_arm_z             : int  -123 -125 -126 -123 -123 -122 -125 -124 -122 -124
##  $ magnet_arm_x            : int  -368 -369 -368 -372 -374 -369 -373 -372 -369 -376
##  $ magnet_arm_y            : int  337 337 344 344 337 342 336 338 341 334
##  $ magnet_arm_z            : int  516 513 513 512 506 513 509 510 518 516
##  $ kurtosis_roll_arm       : num  NA NA NA NA NA NA NA NA NA NA
##  $ kurtosis_picth_arm      : num  NA NA NA NA NA NA NA NA NA NA
##  $ kurtosis_yaw_arm        : num  NA NA NA NA NA NA NA NA NA NA
##  $ skewness_roll_arm       : num  NA NA NA NA NA NA NA NA NA NA
##  $ skewness_pitch_arm      : num  NA NA NA NA NA NA NA NA NA NA
##  $ skewness_yaw_arm        : num  NA NA NA NA NA NA NA NA NA NA
##  $ max_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA
##  $ max_picth_arm           : num  NA NA NA NA NA NA NA NA NA NA
##  $ max_yaw_arm             : int  NA NA NA NA NA NA NA NA NA NA
##  $ min_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA
##  $ min_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA
##  $ min_yaw_arm             : int  NA NA NA NA NA NA NA NA NA NA
##  $ amplitude_roll_arm      : num  NA NA NA NA NA NA NA NA NA NA
##  $ amplitude_pitch_arm     : num  NA NA NA NA NA NA NA NA NA NA
##  $ amplitude_yaw_arm       : int  NA NA NA NA NA NA NA NA NA NA
##  $ roll_dumbbell           : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell          : num  -70.5 -70.6 -70.3 -70.4 -70.4 ...
##  $ yaw_dumbbell            : num  -84.9 -84.7 -85.1 -84.9 -84.9 ...
##  $ kurtosis_roll_dumbbell  : num  NA NA NA NA NA NA NA NA NA NA
##  $ kurtosis_picth_dumbbell : num  NA NA NA NA NA NA NA NA NA NA
##  $ kurtosis_yaw_dumbbell   : logi  NA NA NA NA NA NA ...
##  $ skewness_roll_dumbbell  : num  NA NA NA NA NA NA NA NA NA NA
##  $ skewness_pitch_dumbbell : num  NA NA NA NA NA NA NA NA NA NA
##  $ skewness_yaw_dumbbell   : logi  NA NA NA NA NA NA ...
##  $ max_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA
##  $ max_picth_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA
##  $ max_yaw_dumbbell        : num  NA NA NA NA NA NA NA NA NA NA
##  $ min_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA
##  $ min_pitch_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA
##  $ min_yaw_dumbbell        : num  NA NA NA NA NA NA NA NA NA NA
##  $ amplitude_roll_dumbbell : num  NA NA NA NA NA NA NA NA NA NA
##   [list output truncated]
```

Clean the data - remove identifying columns, zero value columns

After further looking at the data we see that the starting 7 columns are identifying and window columns. Also there are columns that are completely zero and will not contribute to the analysis.

```r
train_data <- train_data[, -c(1:7)]
test_data <- test_data[, -c(1:7)]

dim(train_data)
```

```
## [1] 19622   153
```

```r
### Now select the rows that has column sums greater then zero
train_data <- train_data[, colSums(is.na(train_data)) == 0]
dim(train_data)
```

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

```r
test_data <- test_data[, colSums(is.na(test_data)) == 0]
dim(test_data)
```

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

```r
str(head(train_data,10))
```

```
## 'data.frame':    10 obs. of  53 variables:
##  $ roll_belt           : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45
##  $ pitch_belt          : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17
##  $ yaw_belt            : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4
##  $ total_accel_belt    : int  3 3 3 3 3 3 3 3 3 3
##  $ gyros_belt_x        : num  0 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 0.03
##  $ gyros_belt_y        : num  0 0 0 0 0.02 0 0 0 0 0
##  $ gyros_belt_z        : num  -0.02 -0.02 -0.02 -0.03 -0.02 -0.02 -0.02 -0.02 -0.02 0
##  $ accel_belt_x        : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21
##  $ accel_belt_y        : int  4 4 5 3 2 4 3 4 2 4
##  $ accel_belt_z        : int  22 22 23 21 24 21 21 21 24 22
##  $ magnet_belt_x       : int  -3 -7 -2 -6 -6 0 -4 -2 1 -3
##  $ magnet_belt_y       : int  599 608 600 604 600 603 599 603 602 609
##  $ magnet_belt_z       : int  -313 -311 -305 -310 -302 -312 -311 -313 -312 -308
##  $ roll_arm            : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128
##  $ pitch_arm           : num  22.5 22.5 22.5 22.1 22.1 22 21.9 21.8 21.7 21.6
##  $ yaw_arm             : num  -161 -161 -161 -161 -161 -161 -161 -161 -161 -161
##  $ total_accel_arm     : int  34 34 34 34 34 34 34 34 34 34
##  $ gyros_arm_x         : num  0 0.02 0.02 0.02 0 0.02 0 0.02 0.02 0.02
##  $ gyros_arm_y         : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03
##  $ gyros_arm_z         : num  -0.02 -0.02 -0.02 0.02 0 0 0 0 -0.02 -0.02
##  $ accel_arm_x         : int  -288 -290 -289 -289 -289 -289 -289 -289 -288 -288
##  $ accel_arm_y         : int  109 110 110 111 111 111 111 111 109 110
##  $ accel_arm_z         : int  -123 -125 -126 -123 -123 -122 -125 -124 -122 -124
##  $ magnet_arm_x        : int  -368 -369 -368 -372 -374 -369 -373 -372 -369 -376
##  $ magnet_arm_y        : int  337 337 344 344 337 342 336 338 341 334
##  $ magnet_arm_z        : int  516 513 513 512 506 513 509 510 518 516
##  $ roll_dumbbell       : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell      : num  -70.5 -70.6 -70.3 -70.4 -70.4 ...
##  $ yaw_dumbbell        : num  -84.9 -84.7 -85.1 -84.9 -84.9 ...
##  $ total_accel_dumbbell: int  37 37 37 37 37 37 37 37 37 37
##  $ gyros_dumbbell_x    : num  0 0 0 0 0 0 0 0 0 0
##  $ gyros_dumbbell_y    : num  -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02
##  $ gyros_dumbbell_z    : num  0 0 0 -0.02 0 0 0 0 0 0
##  $ accel_dumbbell_x    : int  -234 -233 -232 -232 -233 -234 -232 -234 -232 -235
##  $ accel_dumbbell_y    : int  47 47 46 48 48 48 47 46 47 48
##  $ accel_dumbbell_z    : int  -271 -269 -270 -269 -270 -269 -270 -272 -269 -270
##  $ magnet_dumbbell_x   : int  -559 -555 -561 -552 -554 -558 -551 -555 -549 -558
##  $ magnet_dumbbell_y   : int  293 296 298 303 292 294 295 300 292 291
##  $ magnet_dumbbell_z   : num  -65 -64 -63 -60 -68 -66 -70 -74 -65 -69
##  $ roll_forearm        : num  28.4 28.3 28.3 28.1 28 27.9 27.9 27.8 27.7 27.7
##  $ pitch_forearm       : num  -63.9 -63.9 -63.9 -63.9 -63.9 -63.9 -63.9 -63.8 -63.8 -63.8
```

```
##  $ yaw_forearm         : num  -153 -153 -152 -152 -152 -152 -152 -152 -152 -152
##  $ total_accel_forearm : int  36 36 36 36 36 36 36 36 36 36
##  $ gyros_forearm_x     : num  0.03 0.02 0.03 0.02 0.02 0.02 0.02 0.02 0.03 0.02
##  $ gyros_forearm_y     : num  0 0 -0.02 -0.02 0 -0.02 0 -0.02 0 0
##  $ gyros_forearm_z     : num  -0.02 -0.02 0 0 -0.02 -0.03 -0.02 0 -0.02 -0.02
##  $ accel_forearm_x     : int  192 192 196 189 189 193 195 193 193 190
##  $ accel_forearm_y     : int  203 203 204 206 206 203 205 205 204 205
##  $ accel_forearm_z     : int  -215 -216 -213 -214 -214 -215 -215 -213 -214 -215
##  $ magnet_forearm_x    : int  -17 -18 -18 -16 -17 -9 -18 -9 -16 -22
##  $ magnet_forearm_y    : num  654 661 658 658 655 660 659 660 653 656
##  $ magnet_forearm_z    : num  476 473 469 469 473 478 470 474 476 473
##  $ classe              : Factor w/ 5 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1
```

Models/Algorithms

Partition the training sample data into training and test(cross validation) sets.

```
# create a partition with the training dataset
train_split  <- createDataPartition(train_data$classe, p=0.7, list=FALSE)
train_df <- train_data[train_split , ]
validate_df  <- train_data[-train_split , ]
## Dimension of Training data
dim(train_df)
```

```
## [1] 13737    53
```

```
## Dimensions of test/cross validation data
dim(validate_df)
```

```
## [1] 5885   53
```

Decision Tree Algorithm

Train the model on the train data

```
# install and load the rpart & plotting library

set.seed(117)
model_file <- "model_dt.RData"
if (!file.exists(model_file )) {
  ##How to avoid overfitting? By changing the minbucket size
  model_dt<- rpart(classe ~ ., data=train_df, method="class", minbucket=50)
  # Plot the tree using fancyRpartPlot command defined in rpart.plot package
  prp(model_dt)

  save(model_dt, file = model_file)
} else {
    load(file=model_file, verbose = TRUE)

}
```

```
## Loading objects:
##   model_dt
```

```
#model_dt
```

Validate the Decision Tree model and compute the accuracy using confusion matrix

```r
validate_dt <- predict(model_dt, validate_df, type = "class")
# Using coinfusion matrix test the accuracy of the model
dt_cm <- confusionMatrix(validate_dt, validate_df$classe)
print(dt_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1554  249   70  159   49
##          B   42  649  122   43   77
##          C   30  114  707   75   78
##          D   38   70   82  566   70
##          E   10   57   45  121  808
##
## Overall Statistics
##
##                Accuracy : 0.728
##                  95% CI : (0.7164, 0.7393)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6528
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9283   0.5698   0.6891  0.58714   0.7468
## Specificity            0.8749   0.9402   0.9389  0.94717   0.9515
## Pos Pred Value         0.7468   0.6956   0.7042  0.68523   0.7762
## Neg Pred Value         0.9685   0.9011   0.9346  0.92133   0.9434
## Prevalence             0.2845   0.1935   0.1743  0.16381   0.1839
## Detection Rate         0.2641   0.1103   0.1201  0.09618   0.1373
## Detection Prevalence   0.3536   0.1585   0.1706  0.14036   0.1769
## Balanced Accuracy      0.9016   0.7550   0.8140  0.76715   0.8491
```

Generalized Boosted Model

Train the model on the train data

```r
# install and load the rpart & plotting library

set.seed(117)
model_file <- "model_gbm.RData"
if (!file.exists(model_file )) {
  ##xross validation
  cv_gbm <- trainControl(method = "repeatedcv", number = 5, repeats = 1)
  model_gbm  <- train(classe ~ ., data=train_df, method = "gbm",
                   trControl = cv_gbm, verbose = FALSE)
  #model_gbm$finalModel
```

```r
  save(model_gbm, file = model_file)
} else {
    load(file=model_file, verbose = TRUE)
    #model_gbm$finalModel
}
```

```
## Loading objects:
##   model_gbm
```

```r
model_gbm$finalModel
```

```
## A gradient boosted model with multinomial loss function.
## 150 iterations were performed.
## There were 52 predictors of which 52 had non-zero influence.
```

Validate the boosted model and compute the accuracy using confusion matrix

```r
validate_gbm <- predict(model_gbm, newdata=validate_df)
# Using coinfusion matrix test the accuracy of the model
gbm_cm <- confusionMatrix(validate_gbm, validate_df$classe)
print(gbm_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1663   30    0    0    0
##          B    8 1086   23    4   12
##          C    3   23  993   25   11
##          D    0    0    9  932   11
##          E    0    0    1    3 1048
##
## Overall Statistics
##
##                Accuracy : 0.9723
##                  95% CI : (0.9678, 0.9763)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.965
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9934   0.9535   0.9678   0.9668   0.9686
## Specificity            0.9929   0.9901   0.9872   0.9959   0.9992
## Pos Pred Value         0.9823   0.9585   0.9412   0.9790   0.9962
## Neg Pred Value         0.9974   0.9888   0.9932   0.9935   0.9930
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2826   0.1845   0.1687   0.1584   0.1781
## Detection Prevalence   0.2877   0.1925   0.1793   0.1618   0.1788
## Balanced Accuracy      0.9932   0.9718   0.9775   0.9814   0.9839
```

Random Forest Algorithm

Random Forest Algorithm on the training data.

```
set.seed(117)
model_file <- "model_rf.RData"
if (!file.exists(model_file )) {

    model_rf <<- randomForest(classe ~ ., data = train_df, mtry = 3, ntree = 150, do.trace = 25, cv.fol
    save(model_rf, file = model_file)
} else {
    load(file=model_file, verbose = TRUE)

}
```

```
## Loading objects:
##   model_rf
```

```
model_rf
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = train_df, mtry = 3,      ntree = 150, do.trace = 25, cv.f
##                Type of random forest: classification
##                      Number of trees: 150
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 0.68%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3899    4    2    0    1 0.001792115
## B   16 2639    3    0    0 0.007148232
## C    0   18 2376    1    1 0.008347245
## D    1    0   34 2217    0 0.015541741
## E    0    0    5    7 2513 0.004752475
```

Test out model on the validation data

Now that we have a model trained on the train data, Evaluate the algorithm efficiency on the test dataset.

```
# prediction on Test dataset
predict_rf <- predict(model_rf, newdata=validate_df)
rf_cm <- confusionMatrix(predict_rf, validate_df$classe)
print(rf_cm)
```

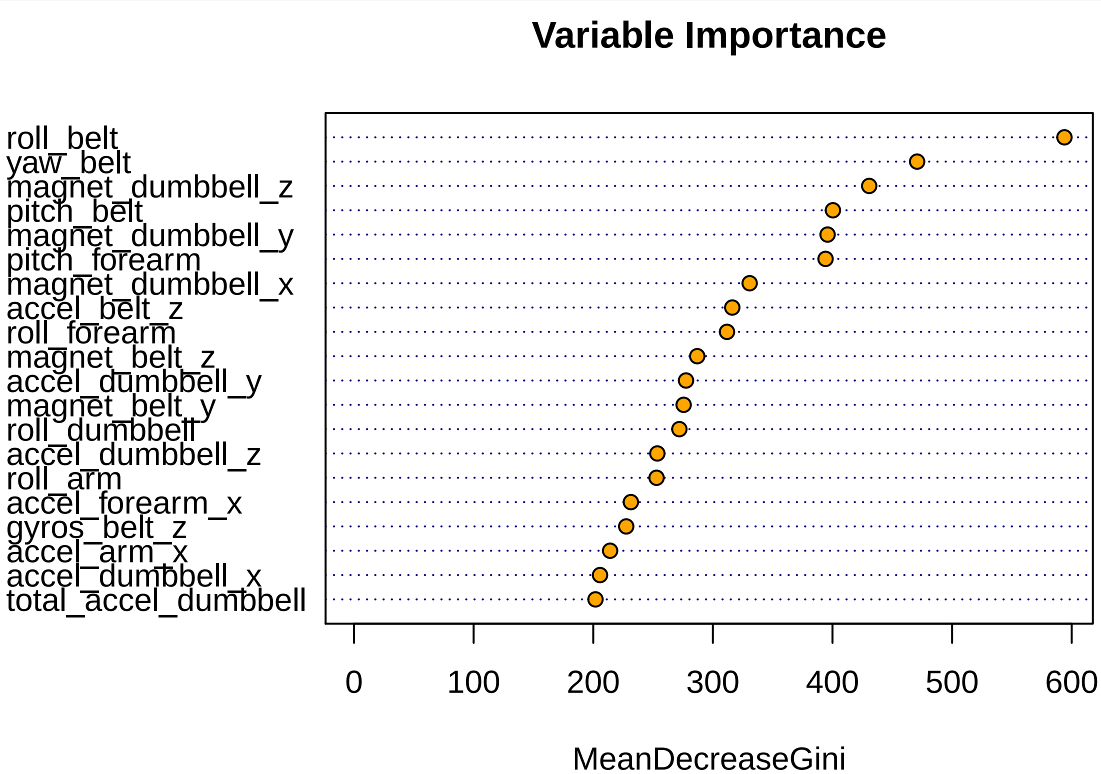
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    2    0    0    0
##          B    0 1136    6    0    0
##          C    0    1 1019    4    0
##          D    0    0    1  960    0
##          E    0    0    0    0 1082
##
## Overall Statistics
##
```

```
##                Accuracy : 0.9976
##                  95% CI : (0.996, 0.9987)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.997
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9974   0.9932   0.9959   1.0000
## Specificity            0.9995   0.9987   0.9990   0.9998   1.0000
## Pos Pred Value         0.9988   0.9947   0.9951   0.9990   1.0000
## Neg Pred Value         1.0000   0.9994   0.9986   0.9992   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1930   0.1732   0.1631   0.1839
## Detection Prevalence   0.2848   0.1941   0.1740   0.1633   0.1839
## Balanced Accuracy      0.9998   0.9981   0.9961   0.9978   1.0000
```

```
# Variable Importance According to Random Forest
rf_var_imp <- as.data.frame(importance(model_rf))
rf_var_imp_sorted <- rf_var_imp[order(rf_var_imp$MeanDecreaseGini),]
head(rf_var_imp_sorted, 20)
```

```
##  [1]  66.88764  85.14107  86.37493  88.26380  98.00043  98.03296 108.11867
##  [8] 108.26045 109.74240 119.51372 120.47223 121.91717 124.00349 128.96581
## [15] 133.36609 133.85392 140.72982 146.75847 148.54954 159.87251
```

```
varImpPlot(model_rf, n.var = 20, sort = TRUE, main = "Variable Importance", lcolor = "navyblue", bg = "
```



Comparing Accuracies

```
dt_accuracy <- dt_cm$overall[1]
gbm_accuracy <- gbm_cm$overall[1]
rf_accuracy <- rf_cm$overall[1]

df_accuracy <- data.frame(Algorithm = c("Decision Tree", "Random Forest", "Gradiant Boost Model"), Inde
df_accuracy <- df_accuracy[order(df_accuracy$Accuracy),]
print(df_accuracy)
```

```
##              Algorithm Index  Accuracy
## 1        Decision Tree    dt 0.7279524
## 3 Gradiant Boost Model   gbm 0.9723025
## 2        Random Forest    rf 0.9976211
```

By comparing the models, we see that that Random Forest is the best performing algorithm with 99.74% accuracy.

Errors

In Sampleerror

```
# In sample Error Rate
InSampError.rf <- (1 - 0.9974)*100
InSampError.rf
```

```
## [1] 0.26
```

In Sample error is: 0.6%

Out Of Sampleerror

```
print(model_rf)
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = train_df, mtry = 3,      ntree = 150, do.trace = 25, cv.f
##                Type of random forest: classification
##                      Number of trees: 150
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 0.68%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3899    4    2    0    1 0.001792115
## B   16 2639    3    0    0 0.007148232
## C    0   18 2376    1    1 0.008347245
## D    1    0   34 2217    0 0.015541741
## E    0    0    5    7 2513 0.004752475
```

As you can see from this output, the OOB is 0.68%.

Generate the test Data Output

Now that we have a working model now, predict the classe for the test data supplied along with the exercvise.

Here based on the comparision of the 3 algorithms provided, we are going to predict using the random forest, since it has the highest accuracy.

```r
predict(model_rf, newdata=test_data)
```

```
##  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
```