Practical Machine Learning Project

Sitaram Gautam

Feb 23, 2016

Background

This analysis corresponds to the prediction assignment for Practical Machine Learning course offered by John Hopkin's University through Coursera. The dataset for the assignment is obtained from http://groupware.le. Based on the website, the data is gathered using different sensors while 6 participants exercised in different fashions. The way they exercised are categorized into A, B, C, D, and E; A being the correct way of doing the exercise and B-E including some level of mistakes.The goal of the assignment is to build an algorithm that takes different fields in the dataset and predicts the associated exercise category correctly.

```
suppressMessages(library(caret))
suppressMessages(library(rpart))
suppressMessages(library(xgboost))

set.seed(977)

training_original <- read.csv(
  "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",
  na.strings = c("NA","#DIV/0!"),
  sep = ",",
  header = TRUE
)
testing_original <- read.csv(
  "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",
  na.strings = c("NA","#DIV/0!"),
  sep = ",",
  header = TRUE
)
```

Data cleaning and preprocessing For this analysis, I am going to use 3 data cleaning/preprocessing techniques which are described below:

1. Take only the fields related to exercises which are belt, arm, dumbbell, and forearm

```
# Take only the fields related to exercises which are belt, arm, dumbbell, and forearm
sensorColumns <-
  grep(pattern = "_belt|_arm|_dumbbell|_forearm", names(training_original))
```

```
# take only the columns associated to
training_original <- training_original[, c(sensorColumns,160)]
```

2. Remove columns with NA

```
# remove columns with NA
cols_wo_na <- colSums(is.na(training_original)) == 0
training_original <- training_original[, cols_wo_na]
```

3. Check for features's variance

```r
## check to see if there is any fields with near zero variability.
near_zero_vars <- nearZeroVar(training_original, saveMetrics = TRUE)
near_zero_vars
```

```
##                      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
```

Columns nzv and zeroVar indicate that none of the features left have near-zero/zero variance i.e no need of features to be filtered out.

Data dividing into training and test set 60 % of the original training set is taken as training set for building model and rest as testing set

```r
## Splitting the original training dataset into another training and a testing set.
inTrain <-
  createDataPartition(y = training_original$classe, p = 0.60, list = FALSE)


training <- training_original[inTrain,]
testing <- training_original[-inTrain,]
```

```r
model_fit_dt <- rpart(classe ~ ., data = training, method = "class")
model_fit_dt
```

Building model

```
## n= 11776
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##    1) root 11776 8428 A (0.28 0.19 0.17 0.16 0.18)
##      2) roll_belt< 130.5 10798 7458 A (0.31 0.21 0.19 0.18 0.11)
##        4) pitch_forearm< -33.95 906    7 A (0.99 0.0077 0 0 0) *
##        5) pitch_forearm>=-33.95 9892 7451 A (0.25 0.23 0.21 0.2 0.12)
##         10) magnet_dumbbell_y< 439.5 8385 5996 A (0.28 0.18 0.24 0.19 0.11)
##           20) roll_forearm< 122.5 5214 3082 A (0.41 0.18 0.19 0.17 0.058)
##             40) magnet_dumbbell_z< -25.5 1802  573 A (0.68 0.2 0.019 0.077 0.025)
##               80) roll_forearm>=-136.5 1510  309 A (0.8 0.16 0.019 0.025 0.0046) *
##               81) roll_forearm< -136.5 292  173 B (0.096 0.41 0.017 0.35 0.13) *
##             41) magnet_dumbbell_z>=-25.5 3412 2479 C (0.26 0.17 0.27 0.22 0.076)
##               82) yaw_belt>=169.5 389   47 A (0.88 0.051 0 0.067 0.0026) *
##               83) yaw_belt< 169.5 3023 2090 C (0.19 0.19 0.31 0.23 0.085)
##                166) accel_dumbbell_y>=-40.5 2622 1927 D (0.21 0.21 0.22 0.27 0.092)
##                  332) pitch_belt< -42.95 313   56 B (0.026 0.82 0.099 0.026 0.029) *
##                  333) pitch_belt>=-42.95 2309 1622 D (0.24 0.12 0.24 0.3 0.1)
##                    666) roll_belt>=125.5 571  235 C (0.36 0.03 0.59 0.014 0.0035)
##                     1332) magnet_belt_z< -326.5 173    3 A (0.98 0 0.0058 0 0.012) *
##                     1333) magnet_belt_z>=-326.5 398   63 C (0.095 0.043 0.84 0.02 0) *
##                    667) roll_belt< 125.5 1738 1059 D (0.2 0.15 0.13 0.39 0.13)
```

```
##                       1334) yaw_belt< -87.55 882  640 B (0.25 0.27 0.14 0.19 0.14)
##                         2668) yaw_belt>=-93.25 732  490 B (0.29 0.33 0.16 0.061 0.15)
##                           5336) magnet_forearm_z>=-43.5 330  135 A (0.59 0.2 0.058 0.067 0.088) *
##                           5337) magnet_forearm_z< -43.5 402  225 B (0.045 0.44 0.25 0.057 0.21) *
##                         2669) yaw_belt< -93.25 150   28 D (0.067 0 0.04 0.81 0.08) *
##                       1335) yaw_belt>=-87.55 856  344 D (0.14 0.03 0.11 0.6 0.12)
##                         2670) yaw_arm< -102.85 92    0 A (1 0 0 0 0) *
##                         2671) yaw_arm>=-102.85 764  252 D (0.034 0.034 0.13 0.67 0.14) *
##                  167) accel_dumbbell_y< -40.5 401   56 C (0.01 0.05 0.86 0.035 0.045) *
##             21) roll_forearm>=122.5 3171 2138 C (0.081 0.18 0.33 0.23 0.18)
##               42) accel_forearm_x>=-108.5 2255 1425 C (0.088 0.22 0.37 0.11 0.22)
##                 84) magnet_forearm_z< -294.5 183   32 A (0.83 0.15 0 0.022 0) *
##                 85) magnet_forearm_z>=-294.5 2072 1242 C (0.023 0.22 0.4 0.12 0.24)
##                  170) magnet_dumbbell_y< 261.5 1052  439 C (0.029 0.18 0.58 0.092 0.12) *
##                  171) magnet_dumbbell_y>=261.5 1020  654 E (0.016 0.27 0.21 0.14 0.36)
##                    342) magnet_arm_y>=186 449  249 B (0.027 0.45 0.31 0.08 0.14) *
##                    343) magnet_arm_y< 186 571  266 E (0.007 0.13 0.13 0.19 0.53) *
##               43) accel_forearm_x< -108.5 916  432 D (0.064 0.079 0.22 0.53 0.11)
##                 86) magnet_arm_y>=296.5 269  103 C (0.041 0.12 0.62 0.17 0.056) *
##                 87) magnet_arm_y< 296.5 647  209 D (0.074 0.063 0.057 0.68 0.13) *
##           11) magnet_dumbbell_y>=439.5 1507  738 B (0.035 0.51 0.036 0.22 0.2)
##             22) total_accel_dumbbell>=5.5 1076  387 B (0.048 0.64 0.05 0.022 0.24)
##               44) roll_belt>=-0.585 902  213 B (0.058 0.76 0.06 0.027 0.092) *
##               45) roll_belt< -0.585 174    0 E (0 0 0 0 1) *
##             23) total_accel_dumbbell< 5.5 431  128 D (0 0.19 0 0.7 0.11) *
##       3) roll_belt>=130.5 978    8 E (0.0082 0 0 0 0.99) *
```

```r
predictions_dt <- predict(model_fit_dt, testing, type = "class")
cm_dt <- confusionMatrix(predictions_dt, testing$classe)
cm_dt
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2023  256   36   63   32
##          B   83  958  220  141  188
##          C   55  158  963  107  105
##          D   53   89  100  896  156
##          E   18   57   49   79  961
##
## Overall Statistics
##
##                Accuracy : 0.7394
##                  95% CI : (0.7295, 0.749)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6693
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
```

```
## Sensitivity            0.9064   0.6311   0.7039   0.6967   0.6664
## Specificity            0.9311   0.9001   0.9344   0.9393   0.9683
## Pos Pred Value         0.8394   0.6025   0.6938   0.6924   0.8256
## Neg Pred Value         0.9616   0.9105   0.9373   0.9405   0.9280
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2578   0.1221   0.1227   0.1142   0.1225
## Detection Prevalence   0.3072   0.2027   0.1769   0.1649   0.1484
## Balanced Accuracy      0.9187   0.7656   0.8192   0.8180   0.8174
```

73.94% overall accuracy. Not the model we want :)

xgboost I am going to try xgboost, another efficient and high performing machine learning algorithm. xgboost works only for numeric data types. So, classe variable in the datasets, which is in factor type, need to be converted to the numeric type.

```r
# convert factor type to numeric
classe_factor <-  training[, "classe"]
classe_numeric <- classe_factor
class_count <- length(levels(classe_factor))
levels(classe_numeric) = 1:class_count


# Remove the classe column from both training and test data sets
training$classe = NULL
testing$classe = NULL


# Change data type of training and testing to matrix
training_matrix = as.matrix(training)
mode(training_matrix) = "numeric"
testing_matrix = as.matrix(testing)
mode(testing_matrix) = "numeric"
class_outcome <- as.matrix(as.integer(classe_numeric) - 1)
```

Apply the xboost model.

```r
model_fit_xgb <-
  xgb.cv(
    param =  list("objective" = "multi:softprob","num_class" = class_count),
    data = training_matrix,
    label = class_outcome,
    nfold = 4,
    nrounds = 100,
    prediction = TRUE,
    verbose = FALSE
  )
predictions_xgb <-
  matrix(
    model_fit_xgb$pred, nrow = length(model_fit_xgb$pred) / class_count, ncol =
      class_count
  )
predictions_xgb <- max.col(predictions_xgb, "last")
confusionMatrix(factor(class_outcome + 1), factor(predictions_xgb))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    2    3    4    5
##          1 3337    7    2    0    2
##          2   18 2255    5    1    0
##          3    0   10 2038    5    1
##          4    0    0   15 1910    5
##          5    0    2    3    7 2153
##
## Overall Statistics
##
##                Accuracy : 0.993
##                  95% CI : (0.9913, 0.9944)
##     No Information Rate : 0.2849
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9911
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9946   0.9916   0.9879   0.9932   0.9963
## Specificity            0.9987   0.9975   0.9984   0.9980   0.9988
## Pos Pred Value         0.9967   0.9895   0.9922   0.9896   0.9945
## Neg Pred Value         0.9979   0.9980   0.9974   0.9987   0.9992
## Prevalence             0.2849   0.1931   0.1752   0.1633   0.1835
## Detection Rate         0.2834   0.1915   0.1731   0.1622   0.1828
## Detection Prevalence   0.2843   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9967   0.9946   0.9931   0.9956   0.9975
```

Overall accuracy is 99%+ which means error rate is less than 1%.

xgboost definitely killed it compared to classification tree! Based on CRAN documentation, xgboost package uses parallel computing on a single machine. It also uses efficient linear model solver.