Practical_ML

Chandar

March 9, 2021

Background

People regularly do exercise 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.

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. The goal of this project 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. The information about the data (Velloso et al., 2013) is available in the following website: http://web.archive.org/web/20161224072740/http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Data reading

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.6.3

## Loading required package: lattice

## Loading required package: ggplot2

## Registered S3 methods overwritten by 'tibble':
##   method     from
##   format.tbl pillar
##   print.tbl  pillar
```

```
train = read.csv(url("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"))
test = read.csv(url("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"))

dim(train)
```

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

```r
dim(test)
```

```
## [1]  20 160
```

```r
summary(train[15:25])     # printing summary for only few variables in the train dataset
```

```
##   skewness_roll_belt skewness_roll_belt.1 skewness_yaw_belt max_roll_belt
##            :19216              :19216              :19216     Min.   :-94.300
##   #DIV/0!  :    9    #DIV/0!  :   32      #DIV/0!:  406      1st Qu.:-88.000
##   0.000000 :    4    0.000000 :    4                          Median : -5.100
##   0.422463 :    2    -2.156553:    3                          Mean   : -6.667
##   -0.003095:    1    -3.072669:    3                          3rd Qu.: 18.500
##   -0.010002:    1    -6.324555:    3                          Max.   :180.000
##   (Other)  :  389    (Other)  :  361                          NA's   :19216
##   max_picth_belt   max_yaw_belt   min_roll_belt     min_pitch_belt
##   Min.   : 3.00          :19216   Min.   :-180.00   Min.   : 0.00
##   1st Qu.: 5.00   -1.1    :   30   1st Qu.: -88.40   1st Qu.: 3.00
##   Median :18.00   -1.4    :   29   Median :  -7.85   Median :16.00
##   Mean   :12.92   -1.2    :   26   Mean   : -10.44   Mean   :10.76
##   3rd Qu.:19.00   -0.9    :   24   3rd Qu.:   9.05   3rd Qu.:17.00
##   Max.   :30.00   -1.3    :   22   Max.   : 173.00   Max.   :23.00
##   NA's   :19216   (Other):  275   NA's   :19216     NA's   :19216
##    min_yaw_belt   amplitude_roll_belt amplitude_pitch_belt
##          :19216   Min.   :  0.000     Min.   : 0.000
##   -1.1    :   30   1st Qu.:  0.300     1st Qu.: 1.000
##   -1.4    :   29   Median :  1.000     Median : 1.000
##   -1.2    :   26   Mean   :  3.769     Mean   : 2.167
##   -0.9    :   24   3rd Qu.:  2.083     3rd Qu.: 2.000
##   -1.3    :   22   Max.   :360.000     Max.   :12.000
##   (Other):  275   NA's   :19216       NA's   :19216
```

The given train data contains 19622 observations, while the test data contains 20 observations While we build a predictive model with this data, we need to split the train data into two set - training set (70%) and validation set (30%), so that overfitting issue will not occur.

Splitting Train set into two - training set and validation set

```r
set.seed(123) # setting seed to make sure the reproducibility of the result

train_index <- createDataPartition(y = train$classe, p = 0.70, list = FALSE)
train_set <- train[train_index,]
valid_set <- train[-train_index,]

dim(train)
```

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

```r
dim(train_set)
```

```
## [1] 13737   160
```

```
dim(valid_set)
```

```
## [1] 5885  160
```

Data Cleaning Process

Going by the summary of the data, we can see that most of the variables are having NAs and blanks for more number of observations, which will not be helpful for modeling and prediction. Hence, we need to exclude those variables having more number of NA's and blanks before we get into modeling part.

```
# for instance, we can check the summary of a variable 'max_picth_belt'
summary(train_set[19])
```

```
##  max_picth_belt
##  Min.   : 3.00
##  1st Qu.: 5.00
##  Median :18.00
##  Mean   :12.88
##  3rd Qu.:19.00
##  Max.   :30.00
##  NA's   :13463
```

```
mean(is.na(train_set[19]))
```

```
## [1] 0.9800539
```

This clearly indicates that nearly 98% of the cases are having NA's in this variable 'max_picth_belt'. Therefore, we can use sapply() function to ignore those variables having NA's for more than 95% of the cases, so that we can have at least 686 observations (13737*0.05=686.85) for training the model.

```
NA_pct = sapply(train_set, function(x) mean(is.na(x))) > 0.95
table(NA_pct)
```

```
## NA_pct
## FALSE  TRUE
##    93    67
```

```
length(NA_pct[NA_pct=='FALSE'])
```

```
## [1] 93
```

```
# Now we need to remove those variables having NAs for more than 95% of th cases
train_set<- train_set[, NA_pct==FALSE]
valid_set<- valid_set[, NA_pct==FALSE]
dim(train_set)
```

```
## [1] 13737    93
```

```
dim(valid_set)
```

```
## [1] 5885   93
```

We can see that the number of variables has been reduced from 160 to 93 by removing variables containing 95% or more NAs. However, we should remove the number of variables further by checking with the variation in the values of each variable and remove those variables having very least variation. That is, we should remove variables having zero variance using the function "nearZeroVar()" in the "train_set".

```
zero_var <- nearZeroVar(train_set)

train_set_f <- train_set[,-zero_var]
valid_set_f <- valid_set[,-zero_var]

dim(train_set_f)
```

```
## [1] 13737    59
```

```
dim(valid_set_f)
```

```
## [1] 5885   59
```

```
names(train_set_f)
```

```
##  [1] "X"                    "user_name"            "raw_timestamp_part_1"
##  [4] "raw_timestamp_part_2" "cvtd_timestamp"       "num_window"
##  [7] "roll_belt"            "pitch_belt"           "yaw_belt"
## [10] "total_accel_belt"     "gyros_belt_x"         "gyros_belt_y"
## [13] "gyros_belt_z"         "accel_belt_x"         "accel_belt_y"
## [16] "accel_belt_z"         "magnet_belt_x"        "magnet_belt_y"
## [19] "magnet_belt_z"        "roll_arm"             "pitch_arm"
## [22] "yaw_arm"              "total_accel_arm"      "gyros_arm_x"
## [25] "gyros_arm_y"          "gyros_arm_z"          "accel_arm_x"
## [28] "accel_arm_y"          "accel_arm_z"          "magnet_arm_x"
## [31] "magnet_arm_y"         "magnet_arm_z"         "roll_dumbbell"
## [34] "pitch_dumbbell"       "yaw_dumbbell"         "total_accel_dumbbell"
## [37] "gyros_dumbbell_x"     "gyros_dumbbell_y"     "gyros_dumbbell_z"
## [40] "accel_dumbbell_x"     "accel_dumbbell_y"     "accel_dumbbell_z"
## [43] "magnet_dumbbell_x"    "magnet_dumbbell_y"    "magnet_dumbbell_z"
## [46] "roll_forearm"         "pitch_forearm"        "yaw_forearm"
## [49] "total_accel_forearm"  "gyros_forearm_x"      "gyros_forearm_y"
## [52] "gyros_forearm_z"      "accel_forearm_x"      "accel_forearm_y"
## [55] "accel_forearm_z"      "magnet_forearm_x"     "magnet_forearm_y"
## [58] "magnet_forearm_z"     "classe"
```

```
# Also, we need to drop the first 5 variables which are not required for modeling as they are like inde
train_set_final <- train_set_f[,-(1:5)]
valid_set_final <- valid_set_f[,-(1:5)]
```

Now we left with only 54 variables including the dependent variable 'classe'. This cleaned data is a healthy data and can be used for model building. Since the dependent variable 'classe' is a categorical one, we can use one of the following three methods: 'random forest', 'gbm', and 'lda'. In this case, we can try random forest, decision tree, and gbm, and the corresponding results can be compared.

Model Building (Random Forest, Decision Tree, and Gradient Boosting Algorithm)

```
set.seed(1234)

#Random Forest
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.6.3

## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

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

```
rf_fit <- randomForest(classe ~., data=train_set_final, importance=TRUE, method="class")
rf_pred <- predict(rf_fit, newdata = valid_set_final)
cm_rf <- confusionMatrix(rf_pred, valid_set_final$classe)
cm_rf$overall['Accuracy']
```

```
##  Accuracy
## 0.9981308
```

```
rf_pred_test <- predict(rf_fit, newdata=test, type = "class")
rf_pred_test
```

```
##  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
```

```
#Decision Tree
library(rpart)
dt_fit <- rpart(classe ~ ., data=train_set_final, method="class")

library(rattle)
```

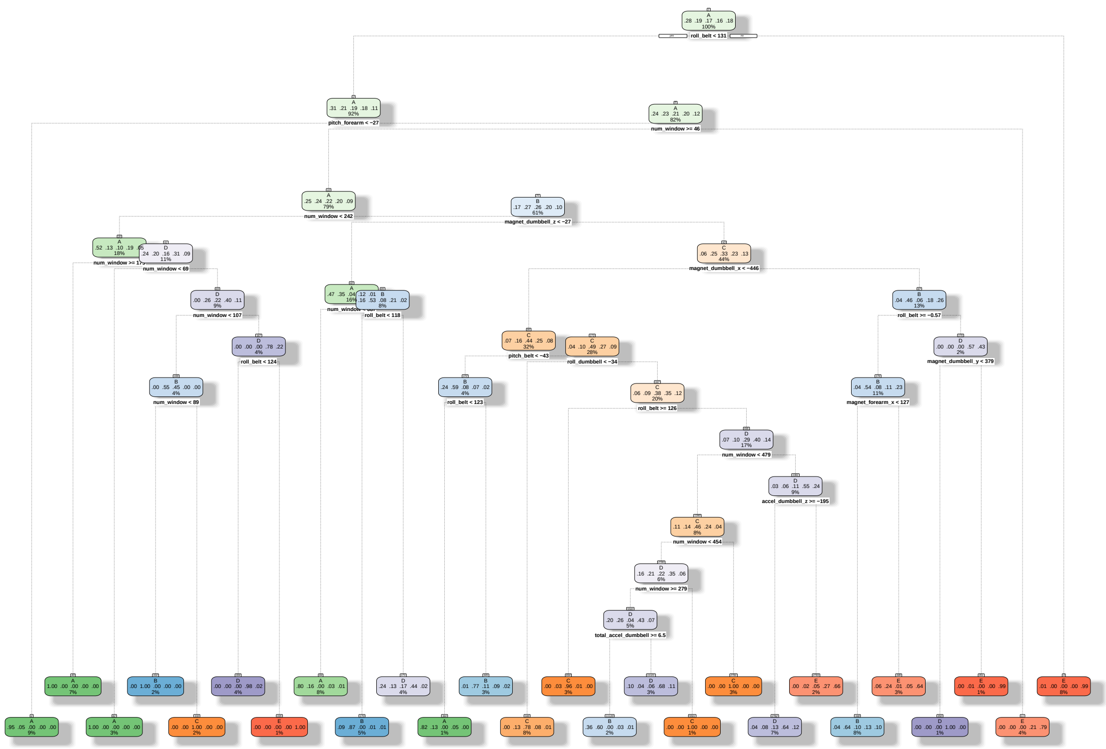
```
## Rattle: A free graphical interface for data science with R.
## Version 5.3.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.

##
## Attaching package: 'rattle'
```

```
## The following object is masked from 'package:randomForest':
##
##     importance
```

```r
fancyRpartPlot(dt_fit)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Rattle 2021−Mar−12 13:05:56 M1028429

```r
dt_pred <- predict(dt_fit, newdata=valid_set_final, type = "class")
head(dt_pred)
```

```
##  3  4  6  8 11 15
##  A  A  A  A  A  A
## Levels: A B C D E
```

```r
cm_dt = confusionMatrix(valid_set_final$classe, dt_pred)
cm_dt
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1459  104    0   92   19
```

```
##          B   86  855   57   81   60
##          C    0   61  856   99   10
##          D   13   75   37  759   80
##          E    2   51    3   86  940
##
## Overall Statistics
##
##                Accuracy : 0.8274
##                  95% CI : (0.8175, 0.8369)
##     No Information Rate : 0.2651
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7823
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9353   0.7461   0.8982   0.6795   0.8476
## Specificity            0.9503   0.9401   0.9655   0.9570   0.9703
## Pos Pred Value         0.8716   0.7507   0.8343   0.7873   0.8688
## Neg Pred Value         0.9760   0.9387   0.9800   0.9273   0.9648
## Prevalence             0.2651   0.1947   0.1619   0.1898   0.1884
## Detection Rate         0.2479   0.1453   0.1455   0.1290   0.1597
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9428   0.8431   0.9319   0.8183   0.9089
```

```
cm_dt$overall['Accuracy']
```

```
##  Accuracy
## 0.8273577
```

```
dt_pred_test <- predict(dt_fit, newdata=test, type = "class")
dt_pred_test
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  B  A  A  A  A  D  D  B  A  A  D  C  E  A  E  E  A  A  B  B
## Levels: A B C D E
```

```
# GBM
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.6.3
```

```
## Loaded gbm 2.1.8
```

```
gbm_fit <- gbm(classe ~.,
               data = train_set_final,
               cv.folds = 3,
               shrinkage = .01,
               n.minobsinnode = 10,
               n.trees = 200, verbose=FALSE)
```

```
## Distribution not specified, assuming multinomial ...

## Warning: Setting 'distribution = "multinomial"' is ill-advised as it is
## currently broken. It exists only for backwards compatibility. Use at your own
## risk.
```

```
gbm_pred <- data.frame(predict(gbm_fit,  newdata=valid_set_final, type="response"))
```

```
## Using 200 trees...
```

```
max(gbm_pred[2,])
```

```
## [1] 0.7630398
```

```
pred_gbm <- as.factor(ifelse(gbm_pred[1]>0.5, 1,
                             ifelse(gbm_pred[2]>0.5, 2,
                                    ifelse(gbm_pred[3]>0.5, 3,
                                           ifelse(gbm_pred[4]>0.5, 4, 5)))))

dat = cbind(valid_actual=valid_set_final$classe, valid_pred = pred_gbm)
cm_boost = confusionMatrix(as.factor(dat[,1]), as.factor(dat[,2]))
cm_boost$overall['Accuracy']
```

```
##  Accuracy
## 0.3039932
```

```
gbm_pred_test <- predict(gbm_fit, newdata=test, type = "response")
```

```
## Using 200 trees...
```

```
test_pred_gbm <- as.factor(ifelse(gbm_pred_test[1]>0.5, 1,
                                  ifelse(gbm_pred_test[2]>0.5, 2,
                                         ifelse(gbm_pred_test[3]>0.5, 3,
                                                ifelse(gbm_pred_test[4]>0.5, 4, 5)))))

test_pred_gbm
```

```
## [1] 5
## Levels: 5
```

Conlcusion:

From the above results, we have 1. Accuracy of the model by random forest = 0.9981308 2. Accuracy of the model by decision tree = 0.8273577 3. Accuracy of the model by gradient boosting algorithm (gbm) = 0.3039932

By comparing the above results, we see that the accuracy for gbm is 0.30, which is very low compared to random forest and decision tree. Also, the accuracy for the random forest model is greater than that of decision tree and gradient boosting algorithm. Hence, we conclude that random forest is best in classifying the labels of the target variable classe (A: Exactly according to the specification, B: Throwing the elbows to the front, C: Lifting the dumbbell only halfway, D: Lowering the dumbbell only halfway, and E: Throwing the hips to the front). Using the final model obtained by each of the three methods, the class has been predicted for the test data of size 20. We can use the one predicted by Random Forest model. The predicted values of classe for the 20 rows of the test dataset by random forest is given as follows:

Predicted values/labels for the test set of size 20 using Random Forest Model:

```
##  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
```

Reference:

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.