Project of Practical Machine Learning

Henry Truong

23/10/2021

Project of Practical Machine Learning

Executive Summary

The goal of this project is to predict the manner in which participants did the exercise. This is the `classe` variable in the training set. The `training` data will be preprocessed to be clean and relevant to the outcomes. After that, `training` data will be divided into sub-data for training and validating. Certain options of training control and methods of preprocessing will be specified. Several models of classification will also be built and the best model will be selected on basis of the validation set. Predicted outcomes of `testing` data will be estimated using this best model.

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, the 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

Data

The training data for this project are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

The test data are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

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

Procedure

Getting and Cleaning Data

Load libraries and data

```r
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```r
training <- read.csv('pml-training.csv')
testing <- read.csv('pml-testing.csv')
```

Check what class each column is
* For character class

```r
isCharVec <- rep(NA, times = 160)
for (i in 1:160) {isCharVec[i] <- is.character(training[1, i])}
sum(isCharVec)
```

```
## [1] 37
```

- For numeric class

```r
isNumVec <- rep(NA, times = 160)
for (i in 1:160) {isNumVec[i] <- is.numeric(training[1, i])}
sum(isNumVec)
```

```
## [1] 123
```

The sum of 123 and 37 is the number of columns of `training` data

Check if columns with numeric class make sense or not
The answer is yes

Check if columns with character class make sense or not The answer is no

```r
names(training)[isCharVec]
```

```
##  [1] "user_name"                "cvtd_timestamp"
##  [3] "new_window"               "kurtosis_roll_belt"
##  [5] "kurtosis_picth_belt"      "kurtosis_yaw_belt"
##  [7] "skewness_roll_belt"       "skewness_roll_belt.1"
##  [9] "skewness_yaw_belt"        "max_yaw_belt"
## [11] "min_yaw_belt"             "amplitude_yaw_belt"
```

```
## [13] "kurtosis_roll_arm"       "kurtosis_picth_arm"
## [15] "kurtosis_yaw_arm"        "skewness_roll_arm"
## [17] "skewness_pitch_arm"      "skewness_yaw_arm"
## [19] "kurtosis_roll_dumbbell"  "kurtosis_picth_dumbbell"
## [21] "kurtosis_yaw_dumbbell"   "skewness_roll_dumbbell"
## [23] "skewness_pitch_dumbbell" "skewness_yaw_dumbbell"
## [25] "max_yaw_dumbbell"        "min_yaw_dumbbell"
## [27] "amplitude_yaw_dumbbell"  "kurtosis_roll_forearm"
## [29] "kurtosis_picth_forearm"  "kurtosis_yaw_forearm"
## [31] "skewness_roll_forearm"   "skewness_pitch_forearm"
## [33] "skewness_yaw_forearm"    "max_yaw_forearm"
## [35] "min_yaw_forearm"         "amplitude_yaw_forearm"
## [37] "classe"
```

- `user_name` should be of character class
- `cvtd_timestamp` should be of date class
- `new_window` should be of factor class
- `classe` should be of factor class
- Other columns with character class should be of numeric class

```
training$cvtd_timestamp <- dmy_hm(training$cvtd_timestamp)
training$new_window <- as.factor(training$new_window)
training$classe <- as.factor(training$classe)
for (i in (1:37)[-c(1, 2, 3, 37)]) {
  training[, isCharVec][, i] <- as.numeric(training[, isCharVec][, i])
  }
```

```
## Warning: NAs introduced by coercion

## Warning: NAs introduced by coercion

## Warning: NAs introduced by coercion

## Warning: NAs introduced by coercion

## Warning: NAs introduced by coercion

## Warning: NAs introduced by coercion

## Warning: NAs introduced by coercion

## Warning: NAs introduced by coercion

## Warning: NAs introduced by coercion

## Warning: NAs introduced by coercion

## Warning: NAs introduced by coercion

## Warning: NAs introduced by coercion

## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

Columns of 'training' are now of appropriate class
Pay attention to `new_window` column

```
table(training$new_window)
```

```
##
##    no   yes
## 19216   406
```

Calculate the number of NAs in each column

```r
numberNAEachCol <- apply(training, 2, function(x) sum(is.na(x)), simplify = T)
```

Extract indices of columns with NA

```r
colIdxWithNA <- which(numberNAEachCol > 0)
length(colIdxWithNA)
```

```
## [1] 100
```

Extract indices of columns without NA

```r
colIdxWithoutNA <- which(numberNAEachCol == 0)
length(colIdxWithNA)
```

```
## [1] 100
```

Pay attention to the number of NAs in columns with NAs

```r
table(numberNAEachCol[colIdxWithNA])
```

```
##
## 19216 19217 19218 19220 19221 19225 19226 19227 19248 19293 19294 19296 19299
##    67     1     1     1     4     1     4     2     2     1     1     2     1
## 19300 19301 19622
##     4     2     6
```

`19216` is the least number of NAs in columns with NAs
There may be a connection between `new_window` column and columns with NAs
Particularly, when `new_window` = 'no', a certain set of columns may have NAs

Let's check

```r
subsetNo <- subset(training, new_window == 'no')
# This subset has 19216 rows and 160 columns
apply(subsetNo, 2, function(x) sum(is.na(x)), simplify = T) -> numberNAEachCol2
colIdxWithNA2 <- which(numberNAEachCol2 > 0)
# There are 100 columns with NAs
```

Check if this set is the same as the previous one

```r
any(colIdxWithNA2 != colIdxWithNA)
```

```
## [1] FALSE
```

The result is `FALSE` so we get the same set of 100 columns with NAs
Check if all elements of these columns are NAs

```r
table(numberNAEachCol2[colIdxWithNA2])
```

```
##
## 19216
##   100
```

Indeed, all of these 100 columns have the same number of NAs which is `19216`
Therefore, when `new_window` = 'no', a set of 100 columns have NAs

Preprocess Data

Note: A predictive model will be built on the assumption of `new_window` = 'no'

Create a subset of 'training' with `new_window` column = 'no' and columns without NAs

```
trainData <- subset(training, new_window == 'no')[, colIdxWithoutNA]
```

Check column names of the subset

```
names(trainData)
```

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

The first 7 columns can be excluded because they are not relevant to `classe` column

```
trainData <- trainData[, -c(1:7)]
```

Split `trainData` into `subTrain` for training and `subValidate` for validating

```
set.seed(1234)
inTrain <- createDataPartition(y = trainData$classe, p = 0.7, list = F)
subTrain <- trainData[inTrain, ]
subValidate <- trainData[-inTrain, ]
```

Set up the training control options when training is performed

```
trainCtrl <- trainControl(method = 'repeatedcv',
                          number = 5,
                          repeats = 3)
```

Building models

A series of classification models will be built using `subTrain` and the performance of each model will be evaluated by `subValidate`

```r
set.seed(1234)
modTree <- train(classe ~ .,
                 data = subTrain,
                 method = 'rpart',
                 trControl = trainCtrl,
                 tuneLength = 5)
predTree <- predict(modTree, subValidate)
conMatTree <- confusionMatrix(predTree, subValidate$classe)
conMatTree$overall
```

Decision Tree

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.5351379      0.3933139      0.5221590      0.5480813      0.2847475
## AccuracyPValue  McnemarPValue
##      0.0000000      0.0000000
```

```r
set.seed(1234)
modRF <- train(classe ~ .,
               data = subTrain,
               method = 'rf',
               trControl = trainCtrl,
               tuneLength = 5)
predRF <- predict(modRF, subValidate)
conMatRF <- confusionMatrix(predRF, subValidate$classe)
conMatRF$overall
```

Random Forest

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9944473      0.9929752      0.9921703      0.9961990      0.2847475
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

```r
set.seed(1234)
modSVM <- train(classe ~ .,
                data = subTrain,
                method = 'svmLinear',
                preProcess = c('center', 'scale', 'nzv'),
                trControl = trainCtrl,
                tuneLength = 5,
                verbose = F)
predSVM <- predict(modSVM, subValidate)
conMatSVM <- confusionMatrix(predSVM, subValidate$classe)
conMatSVM$overall
```

Support Vector Machine

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   7.806698e-01   7.211743e-01   7.697590e-01   7.912957e-01   2.847475e-01
## AccuracyPValue  McnemarPValue
##   0.000000e+00   2.271739e-45
```

```r
set.seed(1234)
modLDA <- train(classe ~ .,
                data = subTrain,
                method = 'lda',
                preProcess = c('center', 'scale', 'nzv'),
                trControl = trainCtrl,
                tuneLength = 5,
                verbose = F)
predLDA <- predict(modLDA, subValidate)
conMatLDA <- confusionMatrix(predLDA, subValidate$classe)
conMatLDA$overall
```

Linear Discriminant Analysis

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   7.020649e-01   6.231541e-01   6.900700e-01   7.138549e-01   2.847475e-01
## AccuracyPValue  McnemarPValue
##   0.000000e+00   6.633020e-59
```

Selecting Best Model

Create a dataframe showing accuracy of each built model

```r
accuracyDF <- data.frame(Model = c('Tree', 'RF', 'SVM', 'LDA'),
                         Accuracy = c(conMatTree$overall[1],
                                      conMatRF$overall[1],
                                      conMatSVM$overall[1],
                                      conMatLDA$overall[1]))
accuracyDF
```

```
##   Model  Accuracy
## 1  Tree 0.5351379
## 2    RF 0.9944473
## 3   SVM 0.7806698
## 4   LDA 0.7020649
```

The ‘modRF’ model will be used to predict the outcomes of ‘testing’ data

Predicting testing data

Transform ‘testing’ data

- `user_name` should be of character class
- `cvtd_timestamp` should be of date class
- `new_window` should be of factor class
- Other columns with character class should be of numeric class

```
testing$cvtd_timestamp <- dmy_hm(testing$cvtd_timestamp)
testing$new_window <- as.factor(testing$new_window)
for (i in (1:37)[-c(1, 2, 3, 37)]) {
  testing[, isCharVec][, i] <- as.numeric(testing[, isCharVec][, i])
}
```

Pay attention to `new_window` column

```
table(testing$new_window)
```

```
##
## no
## 20
```

All values of `new_window` column are 'no'. It will be reasonable to apply `modRF` model on `testing` data because the model was built on the assumption of `new_window` = 0

Subset non-NA columns from `testing` data using `colIdxWithoutNA` from `training`

```
testData <- testing[, colIdxWithoutNA]
```

Check if there is any NA value in new `testData` data

```
sum(sum(is.na(testData)))
```

```
## [1] 0
```

The result is 0 meaning there is no NA

Making prediction Using transformed `testData`

```
testPred <- predict(modRF, testData)
testPred
```

```
##  [1] 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
```