Practical Machine Learning course project

Yiqun HUANG

1/10/2020

Summary

This report is the writeup of Practical Machine Learning course project. The main goal of this project is fitting models to predict the manner that 6 participants performed in the dataset.

The project is developed in the following content

- Overview of the dataset
- Exploratory data analysis and preprocess of dataset
- Model fitting and validation

Overview of the dataset

Using devices is now possible to collect a large amount of data about personal activity. In this dataset, six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions: exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E). The goal of this project is to predict the manner in which they did the exercise, which is the “classe” variable in the training set.

Load dataset and packages will be used in the project

```
library(caret)
```

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

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

```
library(rpart)
library(rpart.plot)
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.
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

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

```
set.seed(113311)

setwd("~/Documents/Coursera R/Pratical Machine Learning/Course Project")
pml.training <- read.csv("~/Documents/Coursera R/Pratical Machine Learning/Course Pro
ject/pml-training.csv")
pml.testing <- read.csv("~/Documents/Coursera R/Pratical Machine Learning/Course Proj
ect/pml-testing.csv")
```

Take a brief look at the training dataset

```
sum(is.na(pml.training))
```

```
## [1] 1287472
```

Creat a partition with the training dataset

```
inTrain <- createDataPartition(y=pml.training$classe, p=0.75, list=FALSE)
training <- pml.training[inTrain,]
testing <- pml.training[-inTrain,]
```

There are 1287472 NA in the training dataset. We need to remove variables that are unbalanced/near zero variance, as well as the variables that are mostly NA.

```
nzv <- nearZeroVar(training)
filteredtraining <- training[, -nzv]
filteredtesting <- testing[, -nzv]

removena <- sapply(filteredtraining, function(x) mean(is.na(x))) > 0.95
TrainSet <- filteredtraining[, removena==FALSE]
TestSet <- filteredtesting[, removena==FALSE]
```

Data processing

Remove the variables are identification information. (participant name)

```
TrainSet <- TrainSet[, -(1:6)]
TestSet <- TestSet[, -(1:6)]
```

Find out highly correlated variables using cor function

```
cor <- cor(TrainSet[ ,-53])
which(cor >0.85, arr.ind = TRUE)
```

```
##                      row col
## roll_belt              1   1
## total_accel_belt       4   1
## accel_belt_y           9   1
## pitch_belt             2   2
## yaw_belt               3   3
## roll_belt              1   4
## total_accel_belt       4   4
## accel_belt_y           9   4
## gyros_belt_x           5   5
## gyros_belt_y           6   6
## gyros_belt_z           7   7
## accel_belt_x           8   8
## magnet_belt_x         11   8
## roll_belt              1   9
## total_accel_belt       4   9
## accel_belt_y           9   9
## accel_belt_z          10  10
## accel_belt_x           8  11
## magnet_belt_x         11  11
## magnet_belt_y         12  12
## magnet_belt_z         13  13
## roll_arm              14  14
## pitch_arm             15  15
## yaw_arm               16  16
## total_accel_arm       17  17
## gyros_arm_x           18  18
## gyros_arm_y           19  19
## gyros_arm_z           20  20
## accel_arm_x           21  21
## accel_arm_y           22  22
## accel_arm_z           23  23
## magnet_arm_x          24  24
## magnet_arm_y          25  25
## magnet_arm_z          26  26
## roll_dumbbell         27  27
## pitch_dumbbell        28  28
## yaw_dumbbell          29  29
## total_accel_dumbbell  30  30
## gyros_dumbbell_x      31  31
## gyros_dumbbell_y      32  32
## gyros_dumbbell_z      33  33
## accel_dumbbell_x      34  34
## accel_dumbbell_y      35  35
## accel_dumbbell_z      36  36
## magnet_dumbbell_x     37  37
## magnet_dumbbell_y     38  38
## magnet_dumbbell_z     39  39
## roll_forearm          40  40
## pitch_forearm         41  41
## yaw_forearm           42  42
## total_accel_forearm   43  43
## gyros_forearm_x       44  44
## gyros_forearm_y       45  45
## gyros_forearm_z       46  46
## accel_forearm_x       47  47
## accel_forearm_y       48  48
```

```
## accel_forearm_z        49   49
## magnet_forearm_x       50   50
## magnet_forearm_y       51   51
## magnet_forearm_z       52   52
```

```
names(TrainSet[c(1, 4, 8, 9, 11)])
```

```
## [1] "roll_belt"         "total_accel_belt" "accel_belt_x"
## [4] "accel_belt_y"      "magnet_belt_x"
```

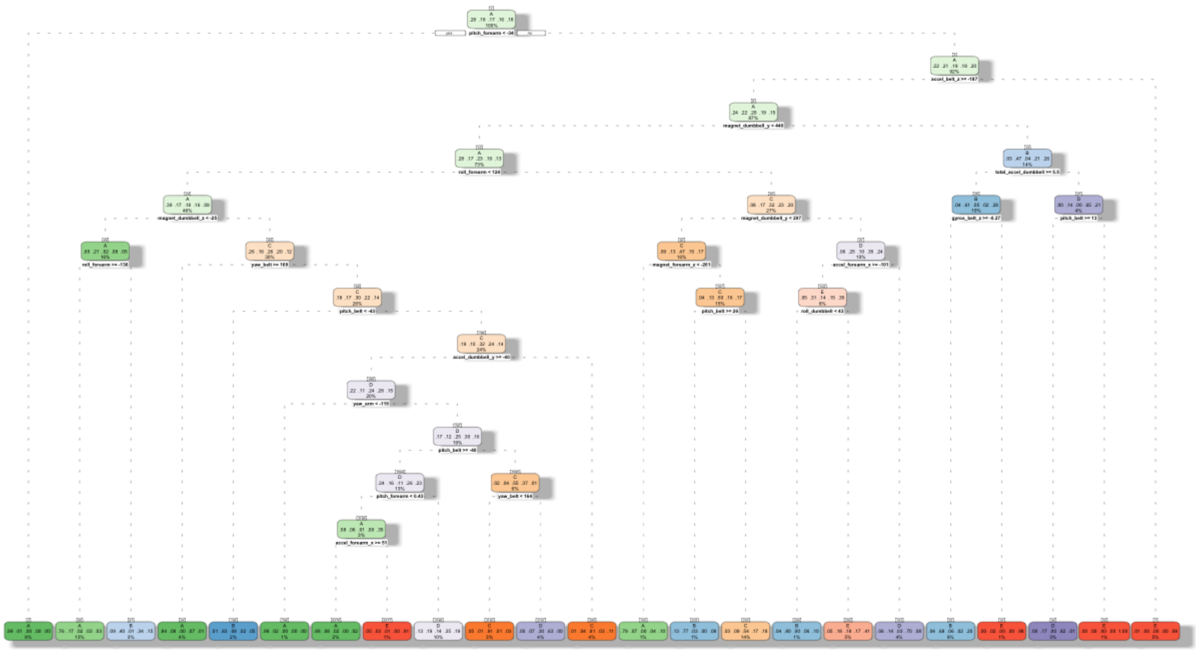
creat the new training and testing dataset omitted the above highly correlated variables

```
TrainSet <- TrainSet[-c(1, 4, 8, 9, 11)]
TestSet <- TestSet[-c(1, 4, 8, 9, 11)]
```

Model fitting

In this project, 3 mostly used classification algorithms in Machine Learnin will be trained. a) Classification trees b) Random forrest c)Boosting trees ###Classification trees

```
FitDT <- rpart(classe ~ ., data= TrainSet, method="class")
fancyRpartPlot(FitDT)
```



Rattle 2020-Jan-29 23:49:40 eileen

Validate the accuracy by using TestSet

```
predictFitDT <- predict(FitDT, TestSet, type = "class")
DecisionTree <- confusionMatrix(predictFitDT, TestSet$classe)
DecisionTree
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1223  142   16   28   31
##          B   49  520   35   55  117
##          C   22   75  631  127  146
##          D   85  151  115  551  106
##          E   16   61   58   43  501
##
## Overall Statistics
##
##                Accuracy : 0.6986
##                  95% CI : (0.6856, 0.7114)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.619
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8767   0.5479   0.7380   0.6853   0.5560
## Specificity            0.9382   0.9353   0.9086   0.8885   0.9555
## Pos Pred Value         0.8493   0.6701   0.6304   0.5466   0.7378
## Neg Pred Value         0.9503   0.8961   0.9426   0.9351   0.9053
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2494   0.1060   0.1287   0.1124   0.1022
## Detection Prevalence   0.2936   0.1582   0.2041   0.2055   0.1385
## Balanced Accuracy      0.9074   0.7416   0.8233   0.7869   0.7558
```

The accuracy is 0.6986, which is quite low and only a little better than guessing.

Random forrest

With the low accuracy of single classfication tree model, Random forrest could be a much powerful model since it contains random decision trees.

```
controlRF <- trainControl(method="cv", number=3, verboseIter=FALSE)
FitRf <- train(classe ~ ., data=TrainSet,
                          method="rf", trControl=controlRF)
FitRf$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 0.69%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 4184    0    0    1    0 0.0002389486
## B   20 2818   10    0    0 0.0105337079
## C    0   16 2549    2    0 0.0070120764
## D    0    0   44 2366    2 0.0190713101
## E    0    0    0    6 2700 0.0022172949
```

Use the TestSet to predict classe~ and validate accuracy

```
predictRf <- predict(FitRf, newdata=TestSet)
confRf <- confusionMatrix(predictRf, TestSet$classe)
confRf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1393    4    0    0    0
##          B    2  942    7    0    0
##          C    0    3  848   12    0
##          D    0    0    0  792    0
##          E    0    0    0    0  901
##
## Overall Statistics
##
##                Accuracy : 0.9943
##                  95% CI : (0.9918, 0.9962)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9928
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9986   0.9926   0.9918   0.9851   1.0000
## Specificity            0.9989   0.9977   0.9963   1.0000   1.0000
## Pos Pred Value         0.9971   0.9905   0.9826   1.0000   1.0000
## Neg Pred Value         0.9994   0.9982   0.9983   0.9971   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2841   0.1921   0.1729   0.1615   0.1837
## Detection Prevalence   0.2849   0.1939   0.1760   0.1615   0.1837
## Balanced Accuracy      0.9987   0.9952   0.9941   0.9925   1.0000
```

The accuracy is very high 0.99. We need to consider over-fitting.

Boosting trees

```
controlGBM <- trainControl(method = "repeatedcv", number = 5, repeats = 1)
modGBM  <- train(classe ~ ., data=TrainSet, method = "gbm", trControl = controlGBM, v
erbose = FALSE)
modGBM$finalModel
```

```
## A gradient boosted model with multinomial loss function.
## 150 iterations were performed.
## There were 47 predictors of which 47 had non-zero influence.
```

```
modGBM
```

```
## Stochastic Gradient Boosting
##
## 14718 samples
##    47 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 11775, 11774, 11773, 11775, 11775
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7337287  0.6624095
##   1                  100      0.8073127  0.7561957
##   1                  150      0.8458365  0.8048743
##   2                   50      0.8495062  0.8093404
##   2                  100      0.9030445  0.8772764
##   2                  150      0.9273683  0.9080790
##   3                   50      0.8919019  0.8631368
##   3                  100      0.9362005  0.9192716
##   3                  150      0.9563114  0.9447207
##
## 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.
```

validate the accuracy with GBM

```
predictGBM <- predict(modGBM, newdata=TestSet)
cmGBM <- confusionMatrix(predictGBM, TestSet$classe)
cmGBM
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1380   33    0    2    1
##          B   10  883   25    5    5
##          C    4   26  815   27    6
##          D    1    4   13  764   13
##          E    0    3    2    6  876
## 
## Overall Statistics
##                                           
##                Accuracy : 0.9621          
##                  95% CI : (0.9563, 0.9672)
##     No Information Rate : 0.2845          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.952           
##  Mcnemar's Test P-Value : 0.001983        
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9892   0.9305   0.9532   0.9502   0.9723
## Specificity            0.9897   0.9886   0.9844   0.9924   0.9973
## Pos Pred Value         0.9746   0.9515   0.9282   0.9610   0.9876
## Neg Pred Value         0.9957   0.9834   0.9901   0.9903   0.9938
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2814   0.1801   0.1662   0.1558   0.1786
## Detection Prevalence   0.2887   0.1892   0.1790   0.1621   0.1809
## Balanced Accuracy      0.9895   0.9595   0.9688   0.9713   0.9848
```

The accuracy is 0.9621. Overall, Random Forrest has the highest accuracy.

Apply trained models to testing dataset

```
quiz <- predict(FitRf, newdata = pml.testing)
quiz
```

```
##  [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
```

```
quiz1 <- predict(modGBM, newdata = pml.testing)
quiz1
```

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

Case 6 has different result on different models.