Peer-graded Assignment: Prediction Assignment

Chengquan Li

01/23/2016

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, your 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. 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

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

Load data

First we should check the availability of data files. If those two data files were saved in local folder, we will read them into memory, otherwise we need get the data from website.

```r
trainingfile<-"pml-training.csv"
testingfile<-"pml-testing.csv"

if(!(file.exists(trainingfile)))
  {
        download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",
                      trainingfile, method = "curl")
        data_training<-read.csv(trainingfile)
        write.csv(data_training,trainingfile)
  } else
  {
          data_training<-read.csv(trainingfile)
  }

if(!(file.exists(testingfile)))
  {
        download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",
                      tesingfile, method = "curl")
        data_testing<-read.csv(testingfile)
        write.csv(data_testing,testingfile)
  } else
  {
          data_testing<-read.csv(testingfile)
  }


# here we set the seed for reproducible analysis.
set.seed(12345)
```

Cleaning Data

At first, let's check the size of data set data_training:

```
str(data_training, list.len = 20)
```

```
## 'data.frame':    19622 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      : Factor w/ 397 levels "","-0.016850",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_belt     : Factor w/ 317 levels "","-0.021887",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt      : Factor w/ 395 levels "","-0.003095",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt.1    : Factor w/ 338 levels "","-0.005928",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ 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            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##   [list output truncated]
```

The training dataset has 160 variables with 19622 observations (rows). We also noticed that there are some "NA" values in cols of this dataset. For the training and prediction, we don't want to deal with those NA data. But first let's check what's the ratio of NA data in those cols. We define a function to check "NA" ratio

```
NARatio<-function(colname){
        NAsum<-sum(is.na(data_training[,colname]))/19622
        return(NAsum)
        }


data_NARatio<-sapply(colnames(data_training),NARatio)

levels(as.factor(data_NARatio))
```

```
## [1] "0"                 "0.979308938946081"
```

We can see that the NA value ratio is either 0 (no NA valuse) or a lot (97% are NA values). Since NA value ratio is very high, we can just remove those cols from taining data.

Remove NA cols

```
# Here we define function NAcol to check if the NA value number is greater than 0.

NAcol<-function(colname){
        if(sum(is.na(data_training[,colname]))>0) {return(T)}
        else { return(F)}
}
NAcols<-sapply(colnames(data_training),NAcol)
new_data_training <- data_training[, !NAcols]
#new_data_testing<-data_testing[,!NAcols]

str(new_data_training,list.len = 20)
```

```
## 'data.frame':    19622 obs. of  93 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 13
##  $ raw_timestamp_part_2    : int  788290 808298 820366 120339 196328 304277 368296 440390 484323 48443
##  $ 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      : Factor w/ 397 levels "","-0.016850",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_belt     : Factor w/ 317 levels "","-0.021887",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt      : Factor w/ 395 levels "","-0.003095",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt.1    : Factor w/ 338 levels "","-0.005928",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_yaw_belt            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ min_yaw_belt            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ amplitude_yaw_belt      : Factor w/ 4 levels "","#DIV/0!","0.00",..: 1 1 1 1 1 1 1 1 1 1 ...
##   [list output truncated]
```

We also notice that there are cols of timestamps, which we will not use for training. And the first column is just the index, we will remove it too.

Remove timestamps cols

```
timestampcols<-grep("timestamp",colnames(new_data_training))

new_data_training<-new_data_training[,-c(1,timestampcols)]
#new_data_testing<-new_data_testing[,-c(1,timestampcols)]

#dim(new_data_testing)
```

```
dim(new_data_training)
```

```
## [1] 19622    89
```

Remove nearly zero covariates

We need check if there is any variable which has no variability at all using the function “nearZeroVar”

```
library(caret)
```

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

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

```
nZV<-nearZeroVar(new_data_training)

new_data_training2<-new_data_training[,-nZV]

#new_data_testing2<-new_data_testing[,-nZV]

#dim(new_data_testing2);dim(new_data_training2)

str(new_data_training2,list.len = 20)
```

```
## 'data.frame':    19622 obs. of  55 variables:
##  $ user_name           : Factor w/ 6 levels "adelmo","carlitos",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ 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 ...
##  $ 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 ...
##   [list output truncated]
```

After removing those near zero covariates, we can see that the col numbers reduced down to 55 from 89. We also see that those cols starting with “#DIV/0!” were removed too. We will use “new_data_training2” and “new_data_testing2” for next model training, testing and predicting.

```
final_train<-new_data_training2
final_predict<-data_testing  #here we use the original testing data for predicting

final_train$classe<-as.factor(final_train$classe)
```

Model training and testing

I found out that with so many rows (19622 observations), train() function in caret package with “rf” or “gbm” method would take too long time for training. Instead, we will use function “randomForest()” and “C5.0()” to do the training and predicting later.

Model training

As usual, we split the data into training and tesing dataset using createDataPartition() with p=0.75. We have 75% data for training and 25% for model testing.

```
inTrain<-createDataPartition(y=final_train$classe,p=0.75,list=FALSE)

final_training_data<-final_train[inTrain,]

final_testing_data<-final_train[-inTrain,]
```

Insead of calling train() function in caret, we will directly use randomForest(), which is much faster. And we will use C5.0 instead of GBM in our data training process.

In randomForest() function, tree number ntree will set to 50, 100 and 500. We can see that the accuracy between ntree=50 and 500 are very close.

```
library(optimbase)
```

```
## Loading required package: Matrix
```

```
library(randomForest)
```

```
## randomForest 4.6-12
```

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

```
##
## Attaching package: 'randomForest'
```

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

```
library(C50)

start <- proc.time()

rf50.fit<-randomForest(classe~.,data=final_training_data,ntree=50)
print(proc.time() - start)
```

```
##    user  system elapsed
##   7.075   0.128   7.254
```

```
start <- proc.time()

rf200.fit<-randomForest(classe~.,data=final_training_data,ntree=200)
print(proc.time() - start)
```

```
##    user  system elapsed
##  25.634   0.429  26.305
```

```
rf500.fit<-randomForest(classe~.,data=final_training_data,ntree=500)
print(proc.time() - start)
```

```
##    user  system elapsed
##  87.406   1.975  90.112
```

```
start <- proc.time()
c50.fit<-C5.0(classe~.,data=final_training_data,rule=TRUE)
print(proc.time() - start)
```

```
##    user  system elapsed
##   4.779   0.150   5.161
```

We can see that speed of randomForest() and C5.0() are really fast, from 5 seconds to 90 seconds with different tree numbers.

Model testing

```
tested.c50<-predict(c50.fit,newdata=final_testing_data)
print(cmc50<-confusionMatrix(tested.c50,final_testing_data$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    4    0    0    0
##          B    0  928   12    1    3
##          C    0   16  842    7    0
##          D    0    1    1  794    1
##          E    0    0    0    2  897
##
## Overall Statistics
##
##                Accuracy : 0.9902
##                  95% CI : (0.987, 0.9928)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9876
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9779   0.9848   0.9876   0.9956
## Specificity            0.9989   0.9960   0.9943   0.9993   0.9995
## Pos Pred Value         0.9971   0.9831   0.9734   0.9962   0.9978
## Neg Pred Value         1.0000   0.9947   0.9968   0.9976   0.9990
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1892   0.1717   0.1619   0.1829
## Detection Prevalence   0.2853   0.1925   0.1764   0.1625   0.1833
## Balanced Accuracy      0.9994   0.9869   0.9896   0.9934   0.9975
```

```r
tested.rf50<-predict(rf50.fit, newdata = final_testing_data)
print(cmrf50<-confusionMatrix(tested.rf50,final_testing_data$classe))
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    3    0    0    0
##          B    0  942    1    0    0
##          C    0    4  854    1    0
##          D    0    0    0  803    1
##          E    0    0    0    0  900
## 
## Overall Statistics
## 
##                Accuracy : 0.998
##                  95% CI : (0.9963, 0.999)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
## 
##                   Kappa : 0.9974
##  Mcnemar's Test P-Value : NA
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9926   0.9988   0.9988   0.9989
## Specificity            0.9991   0.9997   0.9988   0.9998   1.0000
## Pos Pred Value         0.9979   0.9989   0.9942   0.9988   1.0000
## Neg Pred Value         1.0000   0.9982   0.9998   0.9998   0.9998
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1921   0.1741   0.1637   0.1835
## Detection Prevalence   0.2851   0.1923   0.1752   0.1639   0.1835
## Balanced Accuracy      0.9996   0.9962   0.9988   0.9993   0.9994
```

```r
tested.rf200<-predict(rf200.fit, newdata = final_testing_data)
print(cmrf200<-confusionMatrix(tested.rf200,final_testing_data$classe))
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    4    0    0    0
##          B    0  942    0    0    0
##          C    0    3  855    0    0
##          D    0    0    0  804    0
##          E    0    0    0    0  901
## 
## Overall Statistics
## 
##                Accuracy : 0.9986
##                  95% CI : (0.9971, 0.9994)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                   Kappa : 0.9982
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9926   1.0000   1.0000   1.0000
## Specificity            0.9989   1.0000   0.9993   1.0000   1.0000
## Pos Pred Value         0.9971   1.0000   0.9965   1.0000   1.0000
## Neg Pred Value         1.0000   0.9982   1.0000   1.0000   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1921   0.1743   0.1639   0.1837
## Detection Prevalence   0.2853   0.1921   0.1750   0.1639   0.1837
## Balanced Accuracy      0.9994   0.9963   0.9996   1.0000   1.0000
```

```
tested.rf500<-predict(rf500.fit, newdata = final_testing_data)
print(cmrf500<-confusionMatrix(tested.rf500,final_testing_data$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    4    0    0    0
##          B    0  941    1    0    0
##          C    0    4  854    1    0
##          D    0    0    0  803    0
##          E    0    0    0    0  901
##
## Overall Statistics
##
##                Accuracy : 0.998
##                  95% CI : (0.9963, 0.999)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9974
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9916   0.9988   0.9988   1.0000
## Specificity            0.9989   0.9997   0.9988   1.0000   1.0000
## Pos Pred Value         0.9971   0.9989   0.9942   1.0000   1.0000
## Neg Pred Value         1.0000   0.9980   0.9998   0.9998   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1919   0.1741   0.1637   0.1837
## Detection Prevalence   0.2853   0.1921   0.1752   0.1637   0.1837
## Balanced Accuracy      0.9994   0.9957   0.9988   0.9994   1.0000
```

Here the accuray of C5.0 is 0.9902121, for randomForest() are 0.9979608, 0.9985726, 0.9979608 with tree number=50, 200, and 500, respectively. Those number are almost same because we have enough data set for training (75% of 19662 rows)

Predict data

Finally we can predict the output of our testing data using our training models

```
finalpredict.rf50<-as.matrix(predict(rf50.fit,newdata=final_predict))
finalpredict.rf200<-as.matrix(predict(rf200.fit,newdata=final_predict))
finalpredict.rf500<-as.matrix(predict(rf500.fit,newdata=final_predict))
finalpredict.c50<-as.matrix(predict(c50.fit,final_predict))

finalpredict<-cbind(finalpredict.c50,finalpredict.rf50,finalpredict.rf200,finalpredict.rf500)
colnames(finalpredict)<-c("C50","rf 50","rf 200","rf 500")



finalpredict<-transpose(finalpredict)

print(finalpredict)
```

```
##        1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17 
## C50    "B" "A" "B" "A" "A" "E" "D" "C" "A" "A" "B" "C" "B" "A" "E" "E" "A"
## rf 50  "B" "A" "B" "A" "A" "E" "D" "B" "A" "A" "B" "C" "B" "A" "E" "E" "A"
## rf 200 "B" "A" "B" "A" "A" "E" "D" "B" "A" "A" "B" "C" "B" "A" "E" "E" "A"
## rf 500 "B" "A" "B" "A" "A" "E" "D" "B" "A" "A" "B" "C" "B" "A" "E" "E" "A"
##        18  19  20 
## C50    "B" "B" "B"
## rf 50  "B" "B" "B"
## rf 200 "B" "B" "B"
## rf 500 "B" "B" "B"
```

The results of these 4 training models are same.

Session information

Following are R and operation system information I used to generate this analysis

```
## R version 3.3.2 (2016-10-31)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: macOS Sierra 10.12.2
## 
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
## 
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base     
## 
## other attached packages:
## [1] C50_0.1.0-24        randomForest_4.6-12 optimbase_1.0-9    
## [4] Matrix_1.2-7.1      caret_6.0-73        ggplot2_2.2.0      
## [7] lattice_0.20-34    
## 
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.9        formatR_1.4        nloptr_1.0.4      
##  [4] plyr_1.8.4         class_7.3-14       iterators_1.0.8   
##  [7] tools_3.3.2        partykit_1.1-1     digest_0.6.10     
## [10] lme4_1.1-12        evaluate_0.10      tibble_1.2        
## [13] gtable_0.2.0       nlme_3.1-128       mgcv_1.8-15
## [16] foreach_1.4.3      yaml_2.1.13        parallel_3.3.2
## [19] SparseM_1.74       e1071_1.6-7        stringr_1.1.0
## [22] knitr_1.14         MatrixModels_0.4-1 stats4_3.3.2
## [25] rprojroot_1.2      grid_3.3.2         nnet_7.3-12
## [28] survival_2.39-5    rmarkdown_1.3      Formula_1.2-1
## [31] minqa_1.2.4        reshape2_1.4.1     car_2.1-4
## [34] magrittr_1.5       backports_1.0.5    scales_0.4.1
## [37] codetools_0.2-15   ModelMetrics_1.1.0 htmltools_0.3.5
## [40] MASS_7.3-45        splines_3.3.2      assertthat_0.1
## [43] pbkrtest_0.4-6     colorspace_1.2-6   quantreg_5.29
## [46] stringi_1.1.2      lazyeval_0.2.0     munsell_0.4.3
```