Prediction Assignment Writeup

Shengnan Huang

12/23/2017

Background

Nowadays large amount of data about personal activity can be collected relatively cheaply by devices such as Jawbone Up, Nike FuelBand, and Fitbit. With these data, people usually quantify how much of a particular activity they do, while ignore the need for quantifying how well they do it.

In this project, with the data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants who were asked to perform barbell lifts correctly and incorrectly in 5 different ways, I will predict the manner in which they did the exercise using certain machine learning algorithms. The outcome variable is "classe", and we need to choose the appropriate predictors.

The data for this project come from this source: http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har.

Basic Exploratory Data Analyses and Preprocessing

First, we load the data sets and necessary packages

```
setwd("/Users/shengnanhuang/Documents/data/Course 8, week 4/")
training<-read.csv("./pml-training.csv")
testing<-read.csv("./pml-testing.csv")
library(caret)
```

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

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

```
## Warning in as.POSIXlt.POSIXct(Sys.time()): unknown timezone 'zone/tz/2017c.
## 1.0/zoneinfo/America/New_York'
```

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
```

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

```
set.seed(1)
dim(training)
```

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

```
dim(testing)
```

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

We use the data in the training variable to build and test our model, so we need to partition the data set into train and test data sets contained respectively in the training2 and testing2 variables. The data in the testing variable is used to answer the questions in the quiz.

```
inTrain<-createDataPartition(training$classe,p=0.6,list=FALSE)
training2<-training[inTrain,]
testing2<-training[-inTrain,]
dim(training2)
```

```
## [1] 11776   160
```

```
dim(testing2)
```

```
## [1] 7846  160
```

For training2, we find 67 columns contain 90% of NA's.

```
a<-sapply(training2, function(y) sum(length(which(is.na(y)))))
length(a[a/dim(training2)[1]>0.9])
```

```
## [1] 67
```

Also, there are 33 columns contain 90% of blank spaces.

```
b<-sapply(training2, function(y) sum(y==""))
b[is.na(b)]<-0
length(b[b/dim(training2)[1]>0.9])
```

```
## [1] 33
```

We delete these columns, and now we have much smaller number of predictors.

```
c<-Reduce('+',list(a,b))
d<-names(c[c==0])
training3<-training2[d]
names(training3)
```

```
##  [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 should not be predictors and we should delete them. Now we have a data frame with 53 columns among which "classe" is the outcome variable and the other 52 are the predictor variables.

```
training3<-training3[,-(1:7)]
dim(training3)
```

```
## [1] 11776    53
```

Prediction Models Building

Random Forests

For the cross-validation part, we divide the data set training3 into 10 folds. Then, we try a random forests model.

```
fitRF<-train(classe ~.,method="rf",data=training3,trControl=trainControl(method="cv",number=10), ntree=1
```

```
fitRF
```

```
## Random Forest
##
## 11776 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 10598, 10598, 10598, 10599, 10598, 10599, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9889607  0.9860338
##   27    0.9902345  0.9876464
##   52    0.9845447  0.9804496
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

We can see that the accuracy is 0.9902345, therefore the in sample error is 0.0097655. Next, we make prediction on our test data set testing3.

```
testing3<-testing2[d]
testing3<-testing3[,-(1:7)]
predRF<-predict(fitRF, testing3)
confRF<-confusionMatrix(predRF,testing3$classe)
confRF
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2224    8    0    0    0
##          B    4 1502    4    0    1
##          C    3    8 1359   17    2
##          D    0    0    5 1268    5
##          E    1    0    0    1 1434
##
## Overall Statistics
##
##                Accuracy : 0.9925
##                  95% CI : (0.9903, 0.9943)
##     No Information Rate : 0.2845
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9905
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9964   0.9895   0.9934   0.9860   0.9945
## Specificity            0.9986   0.9986   0.9954   0.9985   0.9997
## Pos Pred Value         0.9964   0.9940   0.9784   0.9922   0.9986
## Neg Pred Value         0.9986   0.9975   0.9986   0.9973   0.9988
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2835   0.1914   0.1732   0.1616   0.1828
## Detection Prevalence   0.2845   0.1926   0.1770   0.1629   0.1830
## Balanced Accuracy      0.9975   0.9940   0.9944   0.9922   0.9971
```

We can see that the accuracy is 0.9925, therefore the out of sample error is 0.0075. That implies the random forests model is pretty good.

Generalized Boosted Model

Then we try the generalized boosted model.

```
set.seed(1)
fitGBM<-train(classe ~.,method="gbm",data=training3,trControl=trainControl(method="cv",number=10))
```

```
## Loading required package: survival
##
## Attaching package: 'survival'
## The following object is masked from 'package:caret':
##
##     cluster
## Loading required package: splines
## Loading required package: parallel
## Loaded gbm 2.1.3
```

```
fitGBM
```

```
## Stochastic Gradient Boosting
##
## 11776 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 10598, 10598, 10599, 10598, 10599, 10598, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7510214  0.6843915
##   1                  100      0.8175106  0.7689253
```

```
##   1                  150      0.8515625  0.8121401
##   2                   50      0.8524125  0.8130087
##   2                  100      0.9074401  0.8828642
##   2                  150      0.9298589  0.9112389
##   3                   50      0.8928344  0.8643366
##   3                  100      0.9406435  0.9248910
##   3                  150      0.9602592  0.9497177
##
## 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.
```

We can see that the accuracy is 0.9602592, therefore the in sample error is 0.0397408. Next, we make prediction on our test data set testing3.

```
predGBM<-predict(fitGBM, testing3)
confGBM<-confusionMatrix(predGBM,testing3$classe)
confGBM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2198   42    0    1    0
##          B   15 1426   42    2   10
##          C   12   46 1311   31   14
##          D    4    3   15 1243   22
##          E    3    1    0    9 1396
##
## Overall Statistics
##
##                Accuracy : 0.9653
##                  95% CI : (0.961, 0.9693)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9561
##  Mcnemar's Test P-Value : 1.297e-09
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9848   0.9394   0.9583   0.9666   0.9681
## Specificity            0.9923   0.9891   0.9841   0.9933   0.9980
## Pos Pred Value         0.9808   0.9538   0.9272   0.9658   0.9908
## Neg Pred Value         0.9939   0.9855   0.9911   0.9934   0.9929
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2801   0.1817   0.1671   0.1584   0.1779
## Detection Prevalence   0.2856   0.1905   0.1802   0.1640   0.1796
## Balanced Accuracy      0.9886   0.9642   0.9712   0.9799   0.9830
```

We can see that the accuracy is 0.9653, therefore the out of sample error is 0.0347. That means generalized boosted model is still pretty good even though the random forests model is better.

Prediction On The Test Data Set (variable: testing)

Both models have very good accuracy, so we use both models to make predictions on the variable testing (to answer the questions in the quiz).

Random Forests

```
predRF2<-predict(fitRF, testing)
predRF2
```

```
##  [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
```

Generalized Boosted Model

```
predGBM2<-predict(fitGBM, testing)
predGBM2
```

```
##  [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
```

Interestingly, both models give the same prediction on this test set, and the prediction proves to be correct.