Practical ML Course Project

Di Zhu

2/3/2021

Executive Summary

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 predict the manner in which people did the exercise using the 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. In the training set, the ‘classe’ variable is the outcome and it can be predicted with any of the other variables.

The report describes the process I built my model and my final choice.That model will be applied to the 20 test cases available in the test data set and the predictions will be submitted to the Course Project Prediction Quiz for automated grading.

Set up the environment and load the data

```
library(caret)
```

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

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

```
Training = read.csv('pml-training.csv', header = TRUE, na.strings = c('', 'NA'))
Testing = read.csv('pml-testing.csv', header = TRUE, na.strings = c('', 'NA'))
dim(Training); dim(Testing)
```

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

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

Data wrangling

Both the training and testing data sets has 160 columns, but many of them are blank, most NAs or useless. The data is prepared by the following steps: 1) Remove the first six columns with id information; 2) Remove the columns that mostly are NAs; 3) Convert classe to factors.

```
Training = Training[, -c(1:6)]
Testing = Testing[, -c(1:6)]

Training = Training[, which(colMeans(!is.na(Training)) > 0.95)]
Testing = Testing[, which(colMeans(!is.na(Testing)) > 0.95)]

Training$classe = factor(Training$classe)

dim(Training); dim(Testing)
```

```
## [1] 19622    54
```

```
## [1] 20 54
```

Now both the training and testing data set has 54 columns left, with the last column either the classe or the problem id.

Create a validation data set

The validation data set contains 25% of the data in the Training data set and will be used to compare the performance of candidate models to help select the best one.

```
set.seed(123)
inTrain <- createDataPartition(y = Training$classe,
                               p = 0.75, list = FALSE)
subTrain <- Training[inTrain, ]
Validation <- Training[-inTrain, ]
```

Correlation Analysis

The correlation analysis is used to reduce the number of potential predictors and thus the running time. Basically, two predictors with high correlation have similar effects on the outcome, so only one of them will be considered in our model.

Here the correlation of all the combination of the 53 predictors is calculated. A function is created to format the correlation matrix into a table with three columns (row names, column names and correlation) to facilitate the following analysis. If two predictors with an absolute correlation higher than 0.75, only one of them will be kept in the data set.

```
corrSubTrain = cor(subTrain[, 1:53])

flattenCorrMatrix = function(rmat){
      ut = upper.tri(rmat)
      data.frame(
            row = rownames(rmat)[row(rmat)[ut]],
            column = colnames(rmat)[col(rmat)[ut]],
            r  = rmat[ut]
      )
}

flaCorrTrain = flattenCorrMatrix(corrSubTrain)

selectCol = unique(flaCorrTrain[abs(flaCorrTrain$r) > 0.75, 'column'])
subTrain = subTrain[, !(colnames(subTrain) %in% selectCol)]
Validation = Validation[, !(colnames(Validation) %in% selectCol)]
Testing = Testing[, !(colnames(Testing) %in% selectCol)]
```

With the correlation analysis, 53 predictors are reduced to 32.

Model building

Three methods, including bootstrap aggregating, random forest and generalized boosted modeling, are used to build the model. Each of them is created on the subTrain data set and then apply to the validation data set. The results are evaluated by accuracy, and the model with the highest accuracy will be selected as our final model.

1) Bootstrap aggregating

```
bagCtrl = bagControl(fit = ctreeBag$fit,
                     predict = ctreeBag$pred,
                     aggregate = ctreeBag$aggregate)
treebag = bag(subTrain[, 1:32], subTrain[, 33],
              B = 10, bagControl = bagCtrl)
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
predBag = predict(treebag, newdata = Validation[, 1:32])
confusionMatrix(predBag, Validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1365   11    1    0    0
##          B   14  918   22    2    8
##          C    6   15  824   26    2
##          D    6    3    7  771    4
##          E    4    2    1    5  887
##
## Overall Statistics
##
##                Accuracy : 0.9717
##                  95% CI : (0.9666, 0.9761)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9642
##
##  Mcnemar's Test P-Value : 0.0007257
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9785   0.9673   0.9637   0.9590   0.9845
## Specificity            0.9966   0.9884   0.9879   0.9951   0.9970
## Pos Pred Value         0.9913   0.9523   0.9439   0.9747   0.9867
## Neg Pred Value         0.9915   0.9921   0.9923   0.9920   0.9965
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2783   0.1872   0.1680   0.1572   0.1809
## Detection Prevalence   0.2808   0.1966   0.1780   0.1613   0.1833
## Balanced Accuracy      0.9875   0.9779   0.9758   0.9770   0.9907
```

2) Random forest

```
rfCtrl = trainControl(method = 'cv', number = 3)
rfmod = train(classe ~ ., data = subTrain, method = 'rf',
              trControl = rfCtrl, tuneGrid = expand.grid(.mtry = c(3:6)))
rfmod
```

```
## Random Forest
##
## 14718 samples
##    32 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 9812, 9812, 9812
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   3     0.9938171  0.9921791
##   4     0.9940889  0.9925229
##   5     0.9950401  0.9937263
##   6     0.9955157  0.9943279
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 6.
```

```
predRf = predict(rfmod, newdata = Validation)
confusionMatrix(predRf, Validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    0    0    0    0
##          B    0  949    3    0    0
##          C    0    0  852    3    0
##          D    0    0    0  801    1
##          E    0    0    0    0  900
##
## 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   1.0000   0.9965   0.9963   0.9989
## Specificity            1.0000   0.9992   0.9993   0.9998   1.0000
## Pos Pred Value         1.0000   0.9968   0.9965   0.9988   1.0000
## Neg Pred Value         1.0000   1.0000   0.9993   0.9993   0.9998
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1935   0.1737   0.1633   0.1835
## Detection Prevalence   0.2845   0.1941   0.1743   0.1635   0.1835
## Balanced Accuracy      1.0000   0.9996   0.9979   0.9980   0.9994
```

3) Generalized Boosted Modeling

```
gbmCtrl = trainControl(method = 'cv', number = 3)
gbmmod = train(classe ~ ., data = subTrain, method = 'gbm',
               trControl = gbmCtrl, verbose = FALSE)
gbmmod
```

```
## Stochastic Gradient Boosting
##
## 14718 samples
##    32 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 9812, 9811, 9813
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7223123  0.6473630
##   1                  100      0.7939940  0.7389794
##   1                  150      0.8465141  0.8058575
##   2                   50      0.8680530  0.8329780
##   2                  100      0.9277761  0.9086246
##   2                  150      0.9586221  0.9476508
##   3                   50      0.9205739  0.8994713
##   3                  100      0.9688137  0.9605472
##   3                  150      0.9858681  0.9821237
##
## 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.
```

```
predGbm = predict(gbmmod, newdata = Validation)
confusionMatrix(predGbm, Validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1394    3    0    0    0
##          B    1  937    5    2    2
##          C    0    5  850   15    0
##          D    0    1    0  787    9
##          E    0    3    0    0  890
##
## Overall Statistics
##
##                Accuracy : 0.9906
##                  95% CI : (0.9875, 0.9931)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9881
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9993   0.9874   0.9942   0.9789   0.9878
## Specificity            0.9991   0.9975   0.9951   0.9976   0.9993
## Pos Pred Value         0.9979   0.9894   0.9770   0.9875   0.9966
## Neg Pred Value         0.9997   0.9970   0.9988   0.9959   0.9973
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2843   0.1911   0.1733   0.1605   0.1815
## Detection Prevalence   0.2849   0.1931   0.1774   0.1625   0.1821
## Balanced Accuracy      0.9992   0.9924   0.9946   0.9882   0.9935
```

The accuracy of the three models are 0.9717, 0.9986 and 0.9906, and **the out-of-sample errors are 0.0283, 0.0014 and 0.0094**, so the random forest model will be our final model.

Apply to the testing data set

```
predTest = predict(rfmod, newdata = Testing)
predTest
```

```
##  [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
```