Machine Learning - Weightlifting Quality Classification

RF Analytics - February 2016

Executive Summary

The object of this study is to develop a model to predict the quality of exercise by using data output from popular sportswear exercise recording devices. The goal of this project is to use data from the device accelerometers, attached to belt, forearm, arm and dumbell of 6 participants. The participants were asked to perform barbell lifts correctly and incorrectly in 5 different ways. The resulting model from this study will be able to be used as an instruction aid to assist people performing weight lifting exercises. The resulting output will allow technique modification by users and trainers to more effectively exercise, while at the same time providing value added benefits from using a sportswear exercise recording device.

More information is available from the website here:.[1]
A very detailed paper of this data collection can be viewed here:.[2]

Exploratory Data Analysis - With Method To Tidy Data

Import the data:

```
raw.train <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv", na.string
raw.test  <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",  na.string
dim(raw.train)
```

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

```
dim(raw.test)
```

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

Visual inspection using the *View()* function, shows many variables have no data. It is desirable to filter them out.
There are 6 unique persons in the training data set, having 19,622 observations of 160 variables. The test set is much smaller having only 20 observations of 60 variables.

```
empty.train <- sapply(raw.train, function(x) sum(is.na(x))) # this gives the columns with the  NA's sum
discard.train <- names(empty.train[empty.train > 1000]) # column names  beyond some threshold of NA's
thin.train <- raw.train[ , !(names(raw.train) %in% discard.train)] # discard 100 empty variables
thin.test <-  raw.test[ , !(names(raw.test) %in% discard.train)]
rm(raw.train); rm(raw.test) # remove the original raw DF's to conserve memory
```

Six young healthy participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions:

Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes. Participants were supervised by an experienced weight lifter to make sure the execution complied to the manner they were supposed to simulate. The exercises were performed by six male participants aged between 20-28 years, with little weight lifting experience. Assurance was made that all participants could easily simulate the mistakes in a safe and controlled manner by using a relatively light dumbbell (1.25kg). The variable "classe" is the attribute that will be used to classify the particular catagory of lifting. It is a factor variable with 5 levels.

```
##  Factor w/ 5 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...
```

A table describing the classification of catagories that will be modeled to fit for prediction is shown below.

Classification	Defintion
A	Exactly according to the specification
B	Throwing the elbows to the front
C	Lifting the dumbbell only halfway
D	Lowering the dumbbell only halfway
E	Throwing the hips to the front

Other variables should be discarded as not being contibutors. X is redundant as a row number in the dataframe, time recordings of the measurement also do not affect *how well they do it*. The variables with the word window in them also do not contribute. These will be manually removed.

Divide data into training and hold out sets

This data set has a large number of observations. With intention of using the Random Forest Model, and its ability to decorrelate variables without overfitting, a slightly agressive portion of the data was used for the training set, 80%. This still allows nearly 4,000 observations to validate errors on the fitted model.

```
train.input <- createDataPartition(thin.train$classe, p = 0.8, list = FALSE)  # 80% set for training -
choochootrain <-   thin.train[train.input, ]
holdoutset <- thin.train[-train.input, ]
```

Model Development

From the class notes and ISLR$_3$, it was clear that simple classification trees and bagging are not as accurate as the Random Forest Model. By intent, we will use minimum classification error to help otpimize the **mtry** parameter of the Random Forest Model. An inherent function of the Random Forest Model, is that it has a built in method of cross validation to prevent over fitting.

The training data was split into a training set for model development, and a *hold out* or validation set.$_4$ The model will be fit using the training data. The resulting model will be applied to the *hold out* set to validate the model accuracy and allow a prediction of the error rate when applied to the test data. The actual test data set is small, only 20 observations, so confidence in the accuracy of a model needs to be established prior to application to the test set.

For the Random Forest Model, it is recommended that m=sqrt(p) for best out of sample error$_5$. For this model, 52 variables result in a value of 7.2. We will try iterations of 6, 7, 8, and 9 predictors, chosen at random (by Random Forest), for each split. This method allows different levels of decorrelating of the trees , thus making the average of the resulting trees less variable and hence more reliable. This results in reduction in test error and out of sample error$_5$. From the results we will pick the best value for the **mtry** parameter with the lowest misclassification error, to apply to the hold out set. The table listed below summarizes these results.

The number of trees in the model is intentionally set to an odd number, 501, so that ties between variable selection for each tree are less related. Manual testing on this data shows that the error rate stabilizes with this tree number size , shown later in this report, so the it is decided to fix the tree size at 501.

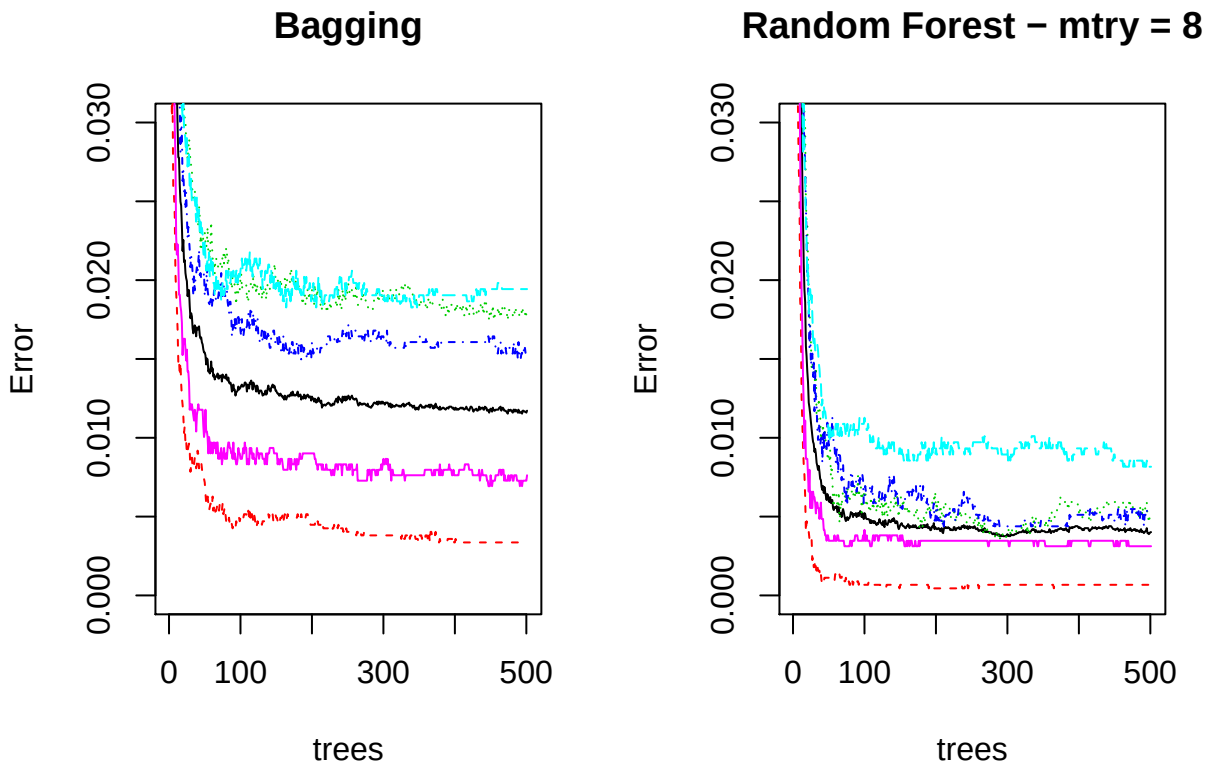
```
set.seed(1963)
bagging           <- randomForest(as.factor(classe) ~., data = choochootrain, method = "rf", mtry=52, nt
weight.model.rf.6 <- randomForest(as.factor(classe) ~., data = choochootrain, method = "rf", mtry=6, nti
weight.model.rf.7 <- randomForest(as.factor(classe) ~., data = choochootrain, method = "rf", mtry=7, nti
weight.model.rf.8 <- randomForest(as.factor(classe) ~., data = choochootrain, method = "rf", mtry=8, nti
weight.model.rf.9 <- randomForest(as.factor(classe) ~., data = choochootrain, method = "rf", mtry=9, nti

rf.error.6 <- round(sum(weight.model.rf.6$confusion[ ,6]), 4)
A6<- sum(weight.model.rf.6$confusion[ c(2,3,4,5),1]); B6 <- sum(weight.model.rf.6$confusion[ c(1,3,4,5)
D6 <- sum(weight.model.rf.6$confusion[ c(1,2,3,5),4]); E6 <- sum(weight.model.rf.6$confusion[ c(1,2,3,4
rf.error.7 <- round(sum(weight.model.rf.7$confusion[ ,6]), 4)
A7<- sum(weight.model.rf.7$confusion[ c(2,3,4,5),1]); B7 <- sum(weight.model.rf.7$confusion[ c(1,3,4,5)
D7 <- sum(weight.model.rf.7$confusion[ c(1,2,3,5),4]); E7 <- sum(weight.model.rf.7$confusion[ c(1,2,3,4
rf.error.8 <- round(sum(weight.model.rf.8$confusion[ ,6]), 4)
A8<- sum(weight.model.rf.8$confusion[ c(2,3,4,5),1]); B8 <- sum(weight.model.rf.8$confusion[ c(1,3,4,5)
D8 <- sum(weight.model.rf.8$confusion[ c(1,2,3,5),4]); E8 <- sum(weight.model.rf.8$confusion[ c(1,2,3,4
rf.error.9 <- round(sum(weight.model.rf.9$confusion[ ,6]), 4)
A9<- sum(weight.model.rf.9$confusion[ c(2,3,4,5),1]); B9 <- sum(weight.model.rf.9$confusion[ c(1,3,4,5)
D9 <- sum(weight.model.rf.9$confusion[ c(1,2,3,5),4]); E9 <- sum(weight.model.rf.9$confusion[ c(1,2,3,4

rf.error.all <- c("",rf.error.6, rf.error.7, rf.error.8, rf.error.9)
class.error.A <- c("",A6,A7,A8,A9)
class.error.B <- c("",B6,B7,B8,B9)
class.error.C <- c("",C6,C7,C8,C9)
class.error.D <- c("",D6,D7,D8,D9)
class.error.E <- c("",E6,E7,E8,E9)

class.table <- data.frame(rbind(rf.error.all, class.error.A, class.error.B, class.error.C, class.error.
colnames(class.table) <- c("", "mtry=6", "mtry=7","mtry=8","mtry=9")

kable(class.table)
```

	mtry=6	mtry=7	mtry=8	mtry=9
rf.error.all	0.023	0.0249	0.0223	0.0226
class.error.A	11	13	11	10
class.error.B	15	13	15	13
class.error.C	30	32	26	29
class.error.D	5	8	7	7
class.error.E	3	5	4	4

In Random Forests, there is no need for cross-validation or a separate test set to get an unbiased estimate of the test set error. It is estimated internally during the run. There is however, value in estimating the predicted misclassification error rate that will be subjected to a test set, as insight to performance.

Random Forest does such a good job of decorrelating variables that all 4 cases of **mtry** would be acceptable for a final fitted model. An mtry = 8, has the lowest classification error and will be used as the model fitted to apply to the holdout set. As a demonstration of the effectiveness of the Random Forest modeling approach, a plot of the fitted model vs the bagging is shown. Random Forest, when tuned such that the number of **mtry** = all the variables (m=p), the result is that of bagging. The **mtry** optimized Random Forest is better able to decorrelate the variables, prevent overfitting and provides a lower misclassification rate.

```
par(mfrow = c(1,2))
  plot(bagging, ylim = c(0,.03),main = "Bagging")
  plot(weight.model.rf.8, ylim = c(0,.03),main = "Random Forest - mtry = 8")
```



The fitted Model is applied to the hold out set to estimate the errors when applied to the test set.

```
predict.ho <- predict(weight.model.rf.8, holdoutset)
confusionMatrix(holdoutset$classe, predict.ho)$table
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1114    2    0    0    0
##          B    3  753    3    0    0
##          C    0    4  680    0    0
##          D    0    0    5  637    1
##          E    0    0    2    0  719
```

```
confusionMatrix(holdoutset$classe, predict.ho)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9949019      0.9935512      0.9921373      0.9968832      0.2847311
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

```
accurate <- round(confusionMatrix(holdoutset$classe, predict.ho)$overall[1],4)*100
```

Using the fitted model, a predicted accuracy of **99.49%** is expected when applied to the test set. This is excellent performance.

The fitted, and validated model is finally applied to the test data to predict "classe" outcome.

```
predict.test <- predict(weight.model.rf.8, thin.test)
predict.test
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```

References:
1. The Groupware website referencing the test method and data: http://groupware.les.inf.puc-rio.br/har.
2. Qualitative Activity Recognition of Weight Lifting Exercises, Velloso, Bulling, Gellersen, Ugulino, and Fuks http://groupware.les.inf.puc-rio.br/public/papers/2013.Velloso.QAR-WLE.pdf.
3. Introducton to Statistical Learning, Gareth, Witten, Hastie, Tibshirani, page 319 - Random Forest improvemnt over other tree methods.
4. Introducton to Statistical Learning, Gareth, Witten, Hastie, Tibshirani, page 176 - Cross validation.
5. Introducton to Statistical Learning, Gareth, Witten, Hastie, Tibshirani, page 319 - optimal value of mtry.