Predicting the Quality of Exercise Routines

John Hopkins University: Practical Machine Learning (Project)

Anthony Cerna

November 5th, 2016

I. Overview

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, the 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).

The goal of this report is to build two predictive classification models using two very well known supervised learning algorithms: **Quadratic Discriminant Analysis** and **Random Forest**. These models will be able to read in new accelerometer data and predict a response label that notifies the user of the quality of their exercise method.

II. Data Description & Importation

i. Data Source

For this assignment, we are provided two data sets. The first is titled the training data set because it contains the true classification response labels. I refer to this one as `labeled.dat` in the my code and it can be downloaded at: * https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv.

The second provided data set is called the test data, which contains 20 new observations that DO NOT have the true classification response label. I refer to this one as `new.dat` in my code and it can be downloaded here: * https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv.

This collected data is part of the **Human Activity Recognition** project. The full source is: (Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. **Qualitative Activity Recognition of Weight Lifting Exercises**. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013.)

The following is a short description of the data that is provided in the author's website: "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).

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. We made sure that all participants could easily simulate the mistakes in a safe and controlled manner by using a relatively light dumbbell (1.25kg).

Read more: http://groupware.les.inf.puc-rio.br/har#dataset#ixzz4PD8xRpmD "

```
labeled.dat <- read.csv("pml-training.csv", row.names = 1)
new.dat <- read.csv("pml-testing.csv", row.names = 1)

dim(labeled.dat)
```

```
## [1] 19622   159
```

```
dim(new.dat)
```

```
## [1]  20 159
```

ii. Variables

As mentioned above, the `labeled.dat` data set contains the true response variable for each observation. This can be found in the **classe** column that species how the exercise was performed:

- Class A: exactly according to the specification
- Class B: throwing the elbows to the front
- Class C: lifting the dumbbell only halfway
- Class D: lowering the dumbbell only halfway
- Class E: throwing the hips to the front

The `new.dat` set, on the other hand, does not contain the true **classe** label. This column is replaced by a **problem_id** variable that is simply the index number of the observation (ranges from 1 to 20). There are a total of 20 new observations that will get assigned a predicted **classe** label by the predictive model in the latter portion of this report.

iii. Partitioning the Labeled Data

The `labeled.dat` data set is split into two parts: training set (70%) and testing set (30%). The 'training' data set will be used to build/train the predictive model and 'testing' data set will be used to check the accuracy of the model. We need to create a testing set that contains the true response variable so that the accuracy of the predictive model can be verified.

```
suppressMessages(library(caret))
set.seed(1000)

inTrain <- createDataPartition(y=labeled.dat$classe, p=0.7, list=FALSE)
training <- labeled.dat[inTrain,]
testing <-  labeled.dat[-inTrain,]

dim(training)
```

```
## [1] 13737   159
```

```
dim(testing)
```

```
## [1] 5885  159
```

III. Data Cleaning

The data sets contain a very large number of columns (159)! We need to compress our data and get rid of columns that do not provide valuable information.

i. Removing Identification Variables

The first four columns of the data set should be removed because these are identification labels that does not provide useful information to the model. For example, the **user_name** will not be helpful because we want a predictive model that can be applied to potential new users. The raw time stamps will also not be of any use because any new observations will have a new unique time stamp of when that data was recorded. Therefore, these columns are excluded.

```
names(labeled.dat)[1:4]
```

```
## [1] "user_name"            "raw_timestamp_part_1" "raw_timestamp_part_2"
## [4] "cvtd_timestamp"
```

```
training <- training[,-c(1:4)]
testing <- testing[,-c(1:4)]

ncol(training) #number of columns decreased by 4
```

```
## [1] 155
```

ii. Identification of Near Zero Variance Predictors

The next task is to remove the predictors that have only one unique value (i.e. zero variability in the column) or predictors that have very few unique values relative to the number of observations. The lack of variability within these types of variables means that it they do not provide a predictive model with relavent information on how to distinguish the observations that belong to different **classe** labels. The `nearZeroVar()` function from the `caret package` does just that.

```
nzv <- nearZeroVar(training[,-ncol(training)]) #do not want to remove the classe column
training <- training[,-nzv] #remove near zero variance predictors
testing <- testing[,-nzv]
ncol(training) #number of columns decreased by 54
```

```
## [1] 101
```

iii. Removing Columns with High NA Percentage

It is important to note that the data still contains columns that have a huge number of NA values in them. This also does not provide relavent information for our predictive models. Any column that contains 95% or more NA's will also be discarded:

```r
na.var <- apply(training, 2, function(x) mean(is.na(x))) > 0.95

training <- training[,na.var==FALSE] #make sure classe variable is included
testing <- testing[,na.var==FALSE]

ncol(training) #number of columns decreased by47
```

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

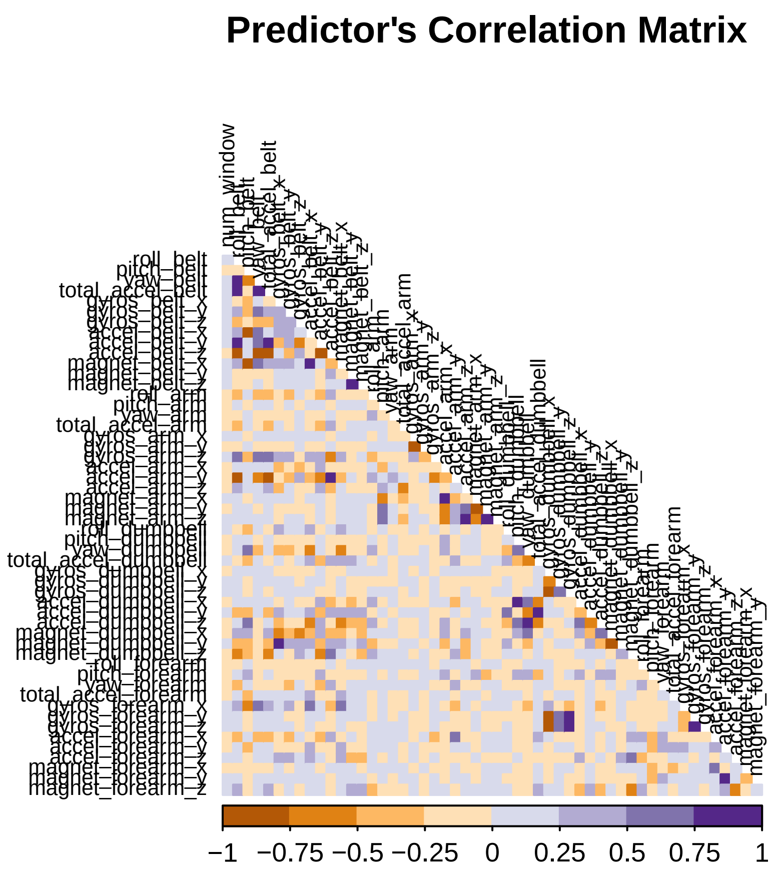
iv. Checking for Independence of Variables

Recall that high correlation amongst predictors can distort the accuracy of a predictive model. Multicollinearity between the variables make it difficult to determine which variables are doing a good job of explaining the variability within a response variable. A useful tool to check thisis a Correlation matrix:

```r
cor.Mat <- cor(training[,-ncol(training)]) #exclude response variable

library(RColorBrewer)
cols <- brewer.pal(n=8, name="PuOr")

library(corrplot)
corrplot(cor.Mat, method="color", type="lower", tl.col="black", tl.cex=.7, col=cols,
         diag=FALSE, title ="Predictor's Correlation Matrix", mar=c(0,0,1,0))
```



The variables that are highly correlated with one another are shown in darker color shades. There are a few with pretty high correlation values. This can be dealt with using Principle Component Analysis. However, most of the variables are not highly correlated with one another. So PCA is excluded here.

After cleaning up our data, we end up with a total of 54 variables (53 predictors + response variable) that will be used for the model building process.

IV. Building Prediction Models: Random Forest

i. Model 1: Quadratic Discriminant Analysis (QDA)

QDA is a very common tool used for classification when the response variable has more than 2 classes. QDA is similar to Linear Discriminant Analysis (LDA) in the sense that the QDA classifier results from assuming that the observations from each class are drawn from a multivariate Gaussian distribution with a class specific mean vector. It then
plugs in estimates for the parameters into Bayes' Theorem in order to perform predictions.

However, unlike LDA, QDA assumes that each class (where K=5 in this case) has its own covariance matrix. LDA is much a much less flexible classifier than QDA, meaning that in theory, it has substantially lower variance. However, Tibshirani et. al. recommends the usage of QDA "if the training set is very large, so that the variance of the classifier is not a major concern, or if the assumption of a common covariance matrix for the K classes is clearly untenable." (James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. New York: Springer, 2013. Print.)

Below, I create a QDA model with the training set and check the accuracy of the model

```r
set.seed(100)
library(MASS)
mod.qda <- qda(classe~., data=training)
mod.qda$prior #prior probabilities of each class
```

```
##         A         B         C         D         E 
## 0.2843416 0.1934920 0.1744195 0.1639368 0.1838101
```

```r
predictQDA.test <- predict(mod.qda, testing)$class
confusionMatrix(predictQDA.test, testing$classe)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1545   58    0    1    0
##          B   72  958   52    5   43
##          C   28  105  967  142   40
##          D   23    4    5  806   30
##          E    6   14    2   10  969
## 
## Overall Statistics
##                                           
##                Accuracy : 0.8912          
##                  95% CI : (0.883, 0.8991) 
##     No Information Rate : 0.2845          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.8627          
##  Mcnemar's Test P-Value : < 2.2e-16       
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9229   0.8411   0.9425   0.8361   0.8956
## Specificity            0.9860   0.9638   0.9352   0.9874   0.9933
## Pos Pred Value         0.9632   0.8478   0.7543   0.9286   0.9680
## Neg Pred Value         0.9699   0.9619   0.9872   0.9685   0.9769
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2625   0.1628   0.1643   0.1370   0.1647
## Detection Prevalence   0.2726   0.1920   0.2178   0.1475   0.1701
## Balanced Accuracy      0.9545   0.9024   0.9388   0.9118   0.9445
```

The QDA Classification model yields an overall accuracy rate of 89.12%. Looking at the confusion matrix, it appears that a lot of the missclassification occured in class C: it has a Positive Prediction Value of only 75%.

ii. Model 2: Random Forest

In this assignment, we are primarily interested in prediction, rather than inference. Random Forest is an excellent model because it has been shown that it has very high accuracy rate. Recall that Random Forest is similar to Bagging in the sense that we build a number of decision trees on bootstrapped training samples.

However, when building these decision trees, each time a split in a tree is considered, *a random sample of m predictors* is chosen as split condidates from the full set of p predictors in our training data.

It is important to use Cross-Validation to determine the optimal value for the parameter `mtry` (i.e. the m value mentioned above). Here we do a 4-fold Cross-Validation.

```
suppressMessages(library(randomForest))
set.seed(100)
control.rf <- trainControl(method="cv", number=4, verboseIter = FALSE)
mod.rf <- train(classe~., data=training, method="rf", trControl=control.rf)
mod.rf$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: 27
##
##         OOB estimate of  error rate: 0.26%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3904    1    0    0    1 0.0005120328
## B    8 2646    4    0    0 0.0045146727
## C    0    7 2388    1    0 0.0033388982
## D    0    0    6 2244    2 0.0035523979
## E    0    0    0    6 2519 0.0023762376
```

Our trained Random Forest model grew 500 decision trees on bootstrapped training samples. The optimal value for `mtry` that was returned from the 4-Fold CV was mtry=27. This final model gives an estimated OOB Error Rate of .26%!

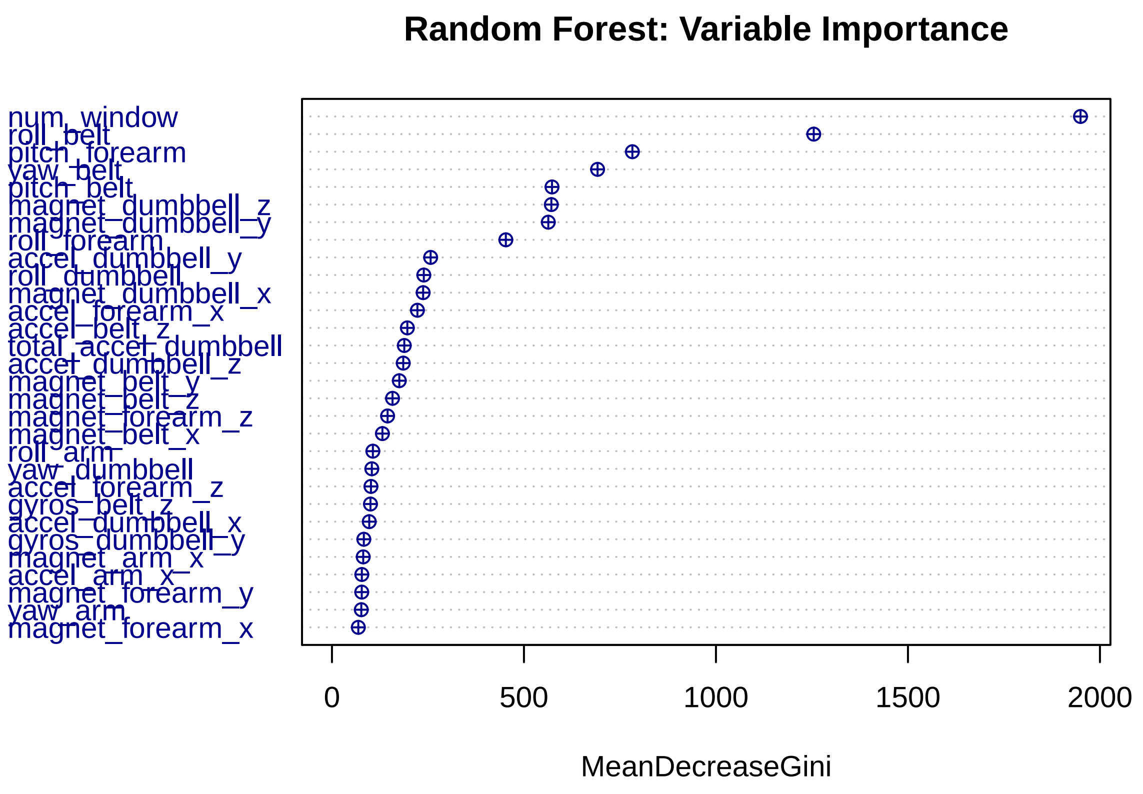
Let's use the created Random Forest model to predict on the "testing" data set and create a confusion matrix to compare the predictions to the actual labels:

```
#Prediction on test data to report Accuracy
predictRF.test <- predict(mod.rf, newdata=testing)
confusionMatrix(predictRF.test, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    6    0    0    0
##          B    0 1132    2    0    0
##          C    0    1 1024    5    0
##          D    0    0    0  959    3
##          E    0    0    0    0 1079
##
## Overall Statistics
##
##                Accuracy : 0.9971
##                  95% CI : (0.9954, 0.9983)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9963
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9939   0.9981   0.9948   0.9972
## Specificity            0.9986   0.9996   0.9988   0.9994   1.0000
## Pos Pred Value         0.9964   0.9982   0.9942   0.9969   1.0000
## Neg Pred Value         1.0000   0.9985   0.9996   0.9990   0.9994
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1924   0.1740   0.1630   0.1833
## Detection Prevalence   0.2855   0.1927   0.1750   0.1635   0.1833
## Balanced Accuracy      0.9993   0.9967   0.9984   0.9971   0.9986
```

The accuracy on the "testing" data is **99%**! Random Forest substantially increases prediction accuracy when compared with other models, however it does this at the expense of interpretability. Fortunately, the `varImpPlot()` function plots the variables in order of decreasing importance in the model. It uses the Average Decrease in the Gini Index as its measure of importance: the larger the decrease in Gini Index, the more important the variable.

```
varImpPlot(mod.rf$finalModel, main="Random Forest: Variable Importance", pch=10, cex=.9, col="dark blue
```



From the above plot, it is clear that **num_window**, **roll_belt**, and **pitch_forearm** are the most important variables in the model

Random Forest might have likely performed better than QDA in terms of accuracy because it did not make any assumption on the distribution of the predictors. ***

V. Predicting Quality of Exercise on New Data

Lastly, we apply the final Random Forest model on 20 new observations (i.e. observations that do not have an actual label `classe`). The model predicts the following labels:

```r
predict(mod.qda, newdata=new.dat)$class
```

```
##  [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
```

```r
predict(mod.rf, newdata = new.dat)
```

```
##  [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
```

In conclusion, it would be best to choose the predicted labels that are outputted by the Random Forest model because the model had a much better prediction accuracy. However, in this case we see that both models predict out the exact same response variable. This concludes the model building process.