Weight Lifting Exercise Classifier Study

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. 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.

In this dataset, 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).

More details regarding this study can be found here: http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har#weight_lifting_exercises

```
buildURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

building <- read.csv(buildURL,
                     na.strings = c("NA", "#DIV/0!", ""),
                     stringsAsFactors = TRUE)

testing <- read.csv(testURL,
                    na.strings = c("NA", "#DIV/0!", ""),
                    stringsAsFactors = TRUE)


building <- building[, -1:-7]
testing <- testing[, -1:-7]
```

Dimension reduction

Due to the large number of variables with a significant proportion of values being NA, these variables contribute little to the prediction algorithms, but come with a significant efficiency cost.

To remove these near-zero-variance variables, we create a function that retains only the variables with the proportion of NA's below a specified threshold. For example, if we specified a threshold of 50%, only variables for which less than 50% of all observations are NA will be retained.

```
retainNAColIndex <- function(argThresh) {
        apply(is.na(building), 2, sum)/dim(building)[1] < argThresh
}
```

By imposing two thresholds at 90% and 10%, we observe that they gave us the same 53 variables. These will be used in our prediction algorithms.

```
sum(retainNAColIndex(.9))
```

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

```
sum(retainNAColIndex(.1))
```

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

```
colsToRetain <- retainNAColIndex(.9)

building <- building[, colsToRetain]
testing <- testing[, colsToRetain]
```

We conduct one final check on the near zero variance covariates, and the test returns no NZV variables. The dataset has been sufficiently cleaned and ready for model training and validation.

```
nearZeroVar(building)
```

```
## integer(0)
```

Training Model

We begin with splitting the dataset into training & validation. We will approach the classification problem with 3 different algorithms, and select one that works best.

```
set.seed(300)

inTrain <- createDataPartition(y = building$classe,
                               p = 0.7,
                               list = FALSE)

training <- building[inTrain, ]
validation <- building[-inTrain, ]
```

Linear Discriminant Analysis

```
modelLDA <- train(data = training,
                  classe ~ .,
                  method = "lda",
                  trControl = trainControl(method = "cv"),
                  preProcess = c("center", "scale"),
                  verbose = FALSE)

cvLDA <- predict(modelLDA, newdata = validation)
confusionMatrix(cvLDA, validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1372  166  111   47   41
##          B   48  718   88   36  176
##          C  112  146  679  119  105
##          D  134   52  128  718   99
##          E    8   57   20   44  661
##
## Overall Statistics
##
##                Accuracy : 0.7048
##                  95% CI : (0.693, 0.7165)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6266
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8196   0.6304   0.6618   0.7448   0.6109
## Specificity            0.9133   0.9267   0.9008   0.9161   0.9731
## Pos Pred Value         0.7899   0.6735   0.5848   0.6348   0.8367
## Neg Pred Value         0.9272   0.9126   0.9265   0.9483   0.9174
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2331   0.1220   0.1154   0.1220   0.1123
## Detection Prevalence   0.2952   0.1811   0.1973   0.1922   0.1342
## Balanced Accuracy      0.8665   0.7785   0.7813   0.8304   0.7920
```

Gradient Boosting Method

```
modelGBM <- train(data = training,
                  classe ~ .,
                  method = "gbm",
                  preProcess <- c("center", "scale"),
                  verbose = FALSE)

cvGBM <- predict(modelGBM, newdata = validation)
confusionMatrix(cvGBM, validation$classe)
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1649   37    0    1    1
##          B   16 1071   30    4   15
##          C    5   31  974   28    6
##          D    2    0   18  923   16
##          E    2    0    4    8 1044
##
## Overall Statistics
##
##                Accuracy : 0.9619
##                  95% CI : (0.9567, 0.9667)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9518
##
##  Mcnemar's Test P-Value : 3.441e-05
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9851   0.9403   0.9493   0.9575   0.9649
## Specificity            0.9907   0.9863   0.9856   0.9927   0.9971
## Pos Pred Value         0.9769   0.9428   0.9330   0.9625   0.9868
## Neg Pred Value         0.9940   0.9857   0.9893   0.9917   0.9921
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2802   0.1820   0.1655   0.1568   0.1774
## Detection Prevalence   0.2868   0.1930   0.1774   0.1630   0.1798
## Balanced Accuracy      0.9879   0.9633   0.9675   0.9751   0.9810
```

Random Forest

```
modelRF <- train(data = training,
                 classe ~ .,
                 method = "rf",
                 trControl = trainControl(method = "cv", number = 4),
                 preProcess = c("center", "scale"),
                 verbose = FALSE)
cvRF <- predict(modelRF, newdata = validation)
confusionMatrix(cvRF, validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1670    5    0    1    0
##          B    2 1130    1    1    1
##          C    1    4 1020    9    2
##          D    0    0    5  953    4
##          E    1    0    0    0 1075
```

```
##
## Overall Statistics
##
##                Accuracy : 0.9937
##                  95% CI : (0.9913, 0.9956)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.992
##
##  Mcnemar's Test P-Value : 0.1239
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9976   0.9921   0.9942   0.9886   0.9935
## Specificity            0.9986   0.9989   0.9967   0.9982   0.9998
## Pos Pred Value         0.9964   0.9956   0.9846   0.9906   0.9991
## Neg Pred Value         0.9990   0.9981   0.9988   0.9978   0.9985
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2838   0.1920   0.1733   0.1619   0.1827
## Detection Prevalence   0.2848   0.1929   0.1760   0.1635   0.1828
## Balanced Accuracy      0.9981   0.9955   0.9954   0.9934   0.9967
```

It appears that random forest is the most accurate algorithm, with an overall accuracy of around 99% on the validation set. It can be reasonably expected the out-of-sample error to be no more than 10%.

Testing

```
predict(modelRF, newdata = testing)
```

```
##  [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
```