Course Project Assignment

Chris

Background

The following description is lifted from the study background provided in the week 4 course project page as part of Coursera's *Practical Machine Learning* learning module.

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. More information is available from the website ***here*** (see the section on the Weight Lifting Exercise Dataset).

Results

Downloading and Loading Data

Our task is to predict the variable `classe` in the training dataset. This variable either represents the manner in which an exercise is being performed, or a type of exercise being performed by a study participant. There is some ambiguity with how the variables in these data should be interpreted as there is no codebook. Regardless, we will use the variables included in the training dataset to predict the `classe` of exercises described in the testing dataset.

We will begin by loading the R packages necessary to complete this analysis, then downloading and reading in the training and testing data.

```
library(tidyverse)
library(caret)
library(rpart)
library(gbm)
library(parallel)
library(doParallel)
cluster <- makeCluster(detectCores() - 2)
registerDoParallel(cluster)
```

We will use the `tidyverse` package to perform any pre-analysis data wrangling. The remainder of the packages here will be used to provide the dependencies necessary for us to train our predictive models.

```
if(!file.exists('train.csv')) {
        url <- 'https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv'
        download.file(url, destfile = 'train.csv')
```

```
}
if(!file.exists('test.csv')) {
        url <- 'https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv'
        download.file(url, destfile = 'test.csv')
}
training <- read.csv('train.csv', na.strings = c('NA', ''))
test <- read.csv('test.csv', na.strings = c('NA', ''))
```

Data Wrangling

We should first inspect the dimensions of these data.

```
dim(training)
```

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

```
dim(test)
```

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

Both training and test datasets contain 160 variables or measurements. We can assume that each row represents a full set of measurements captured while the study participant was performing an exercise. Have a look at the variables encoded in the training dataset. We will only print info for the first fifteen variables but you are welcome to have a look at all the variables by subsetting the `15` with `ncol(training)`

```
str(training, list.len = 15)
```

```
## 'data.frame':    19622 obs. of  160 variables:
##  $ X                       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ user_name               : chr  "carlitos" "carlitos" "carlitos" "carlitos" ...
##  $ raw_timestamp_part_1    : int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 1
##  $ raw_timestamp_part_2    : int  788290 808298 820366 120339 196328 304277 368296 440390 484323 484
##  $ cvtd_timestamp          : chr  "05/12/2011 11:23" "05/12/2011 11:23" "05/12/2011 11:23" "05/12/20
##  $ new_window              : chr  "no" "no" "no" "no" ...
##  $ num_window              : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ roll_belt               : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45 ...
##  $ pitch_belt              : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17 ...
##  $ yaw_belt                : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 ...
##  $ total_accel_belt        : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ kurtosis_roll_belt      : chr  NA NA NA NA ...
##  $ kurtosis_picth_belt     : chr  NA NA NA NA ...
##  $ kurtosis_yaw_belt       : chr  NA NA NA NA ...
##  $ skewness_roll_belt      : chr  NA NA NA NA ...
##    [list output truncated]
```

Some variables in the training set are unlikely to be informative in our predictive model. For instance, the `X` variable just encodes the row number. The `user_name` likely corresponds to the study participant. Other variables like `cvtd_timestamp` have to do with bookkeeping relating to how and when the data were collected and should be removed.

```
training.filt <- training %>%
        dplyr::select(-c(X, user_name, raw_timestamp_part_1, raw_timestamp_part_2,
                         cvtd_timestamp, new_window, num_window))
```

Variance that have zero or near-zero variance are also unlikely to help our models discriminate between different groups. We can go ahead and remove them.

```
training.filt <- training.filt[, -nearZeroVar(training.filt)]
```

You'll also notice that many variables are populated with lots of `NA` values. We will remove variables that contain `NA` values in at least 50% of the total number of rows. This will also help reduce the computational burden of having to train a model with so many different features.

```
training.filt <- training.filt %>%
        dplyr::select(-which(colSums(is.na(training.filt)) / nrow(training.filt) > 0.5))
```

The `classe` variable is the outcome we are interested in predicting. We will change it from a `character` variable to a proper `factor` variable.

```
training.filt$classe <- factor(training.filt$classe)
```

Fitting predictive models

Now that we've cleaned up our training data, we will fit three predictive models using three different learning methods. We will first train a random forest model. Random forests consider the predictions from many separate decision trees, trained using different subsets of bootstrap samples, to generate a single prediction by plurality vote.

We should first list out some parameters for our training algorithm. We would like our training dataset to be resampled through 5-fold cross-validation (as opposed to bootstrapping), and we would also like R to take advantage of multi-threading in order to speed up the training.

With that done, let's go ahead and train our random forest model first.

```
fit.rf <- train(classe ~., data = training.filt, method = 'rf',
                trControl = fitControl, verbose = FALSE)
```

Gradient boosted learning is similar to the random forest technique. Whereas random forest utilizes a parallel combination of decision trees, gradient boosting allows each successive decision tree to iteratively try to minimize the error of the prediction.

```
fit.gbm <- train(classe ~., data = training.filt, method = 'gbm',
                 trControl = fitControl, verbose = FALSE)
```

Model Error Estimates

We will estimate which model will do best on the test dataset by examining the cross-validated errors. The most accurate model will be advanced to the next step, where we will apply it to the test dataset.

```
fit.rf$results
```

```
##   mtry  Accuracy     Kappa  AccuracySD     KappaSD
## 1    2 0.9939863 0.9923926 0.001903606 0.002408284
## 2   27 0.9936805 0.9920061 0.001400622 0.001771544
## 3   52 0.9878198 0.9845920 0.002164742 0.002737028
```

```
fit.gbm$results
```

```
##   shrinkage interaction.depth n.minobsinnode n.trees  Accuracy     Kappa
## 1       0.1                 1             10      50 0.7545092 0.6887850
## 4       0.1                 2             10      50 0.8550597 0.8163339
## 7       0.1                 3             10      50 0.8993979 0.8726131
## 2       0.1                 1             10     100 0.8224945 0.7752948
## 5       0.1                 2             10     100 0.9055645 0.8804900
## 8       0.1                 3             10     100 0.9427682 0.9275738
## 3       0.1                 1             10     150 0.8556211 0.8172638
## 6       0.1                 2             10     150 0.9330338 0.9152683
## 9       0.1                 3             10     150 0.9621339 0.9520888
##    AccuracySD     KappaSD
## 1 0.009943222 0.012978115
## 4 0.010442538 0.013367630
## 7 0.005813000 0.007407003
## 2 0.011111328 0.014110717
## 5 0.005157361 0.006555374
## 8 0.004389019 0.005582925
## 3 0.007499964 0.009542564
## 6 0.005878190 0.007449629
## 9 0.003584932 0.004550402
```

Have a look at the accuracy column in each model. The random forest model seems to have the lowest cross-validated error. We will advance this model to the next step.

Predicting Exercise Classes in Test Set

Now that we have selected our best-performing predictive model, we will apply it to the holdout test set.

```
predict(fit.rf, test)
```

```
##  [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
```

These are the predictions that we will provide as answers to the end-of-course quiz. Thanks for listening, bitchezz!