# Human Activity Recognition: Predicting the Quality of Movements

*Emily Petrie*

*10/20/2018*

## Background

A plethora of wearable technologies have emerged all designed to monitor and report a user's movements. Rarely, however, do these technologies report how well users perform a movement. The purpose of this report is to build a model off of a robust human activity recognition dataset which predicts the quality of a movement (in this case, bicep curls) that a user performs, based on information reported by wearable devices.

## Preparing the Data

Both training and validation dataset were provided. The first step we will take is to partition the training data into yet another training and testing set. The latter dataset will be used to evaluate the efficacy of two different models trained on the former: the first a decision tree model, and the second a random forest model.

```
set.seed(10202019)

#load training data
train <- read.csv("pml-training.csv")

#break out training dataset into additional training and testing subsets
inTrain <- createDataPartition(y = train$classe, p = 0.6, list = F)
subTrain <- train[inTrain, ]
subTest <- train[-inTrain, ]
```

## Selecting Predictors

Next, we will identify which variables will be used to predict for the movement class variable which is the variable of interest. First, we remove variables with little to no variation.

```
nsv <- nearZeroVar(subTrain, saveMetrics = F)
subTrain2 <- subTrain[,-nsv]
```

Next, we will remove variables that will logically not be useful for predicting movement class.

```
subTrain3 <- subTrain2 %>%
      select(-c(X, user_name, raw_timestamp_part_1, raw_timestamp_part_2, cvtd_timestamp))
```

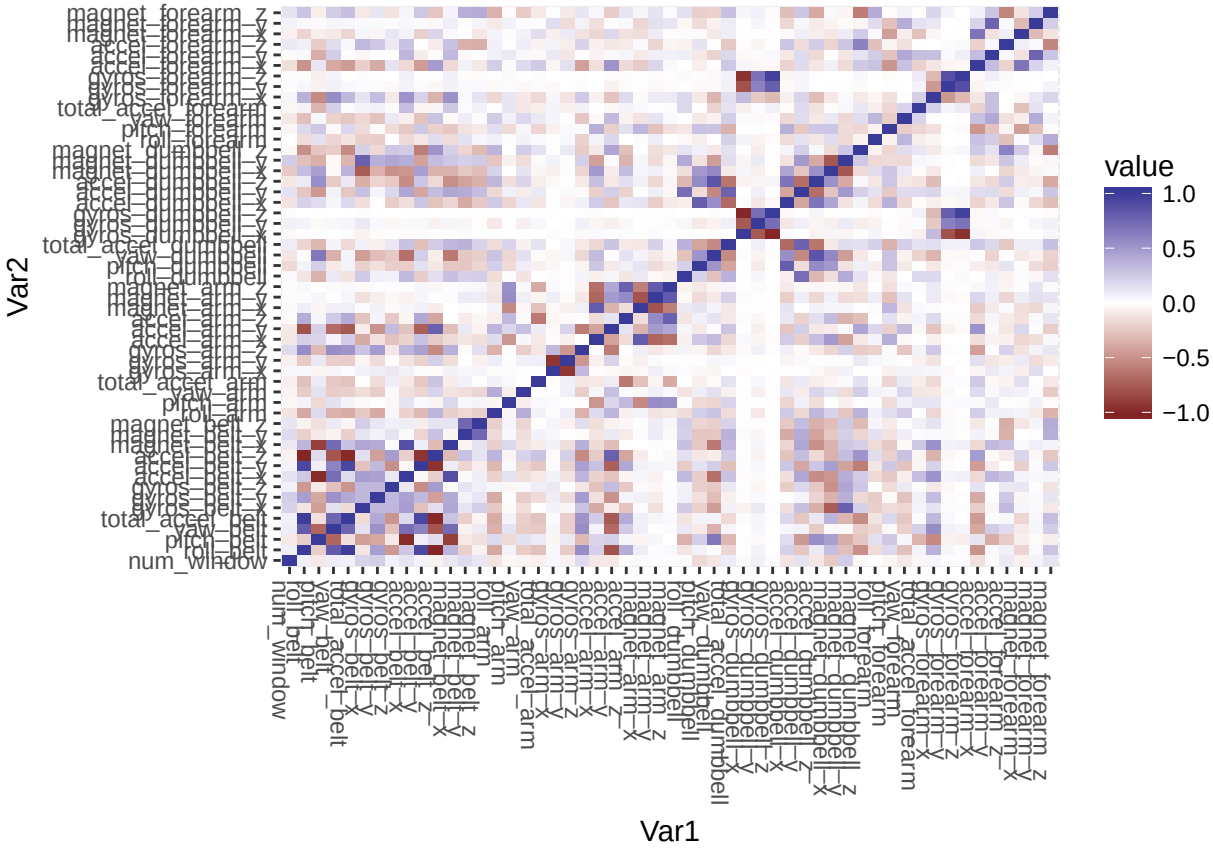
Next, we will remove variables that contain too many NAs (arbitrarily defined as more than 20% of values).

```
remove <- c()
for(i in 1:ncol(subTrain3)){
  if(sum(is.na(subTrain3[,i]))/nrow(subTrain3) >= 0.2){
    remove <- c(remove, colnames(subTrain3)[i])
  }
}
```

```
indx <- match(remove, names(subTrain3))
subTrain4 <- subTrain3[,-indx]
```

Finally, we are going to look at whether or not there are variables in the data that are highly correlated with each other - meaning there is little value in using both to make predictions. Based on the correlation matrix below, there do seem to be some highly correlated pairs.

```
corrs <- cor(subTrain4[sapply(subTrain4, is.numeric)])
corrs_viz <- melt(corrs)
qplot(x = Var1, y = Var2, data = corrs_viz, fill = value, geom = "tile") +
  scale_fill_gradient2(limits = c(-1, 1)) +
  theme(axis.text.x = element_text(angle = -90, hjust = 0))
```




For each pair of highly correlated variables, we will remove the variable with the largest mean absolute correlation (i.e. the one that is most highly correlated with the rest of the dataset).

```
redundant <- findCorrelation(corrs, cutoff = .90)
subTrain5 <- subTrain4[,-redundant]
```

Before proceeding, we must replicate all of the above steps on the testing dataset.

```
subTest2 <- subTest[,-nsv]
subTest3 <- subTest2 %>%
  select(-c(X, user_name, raw_timestamp_part_1, raw_timestamp_part_2, cvtd_timestamp))
indx <- match(remove, names(subTest3))
subTest4 <- subTest3[,-indx]
subTest5 <- subTest4[,-redundant]
```

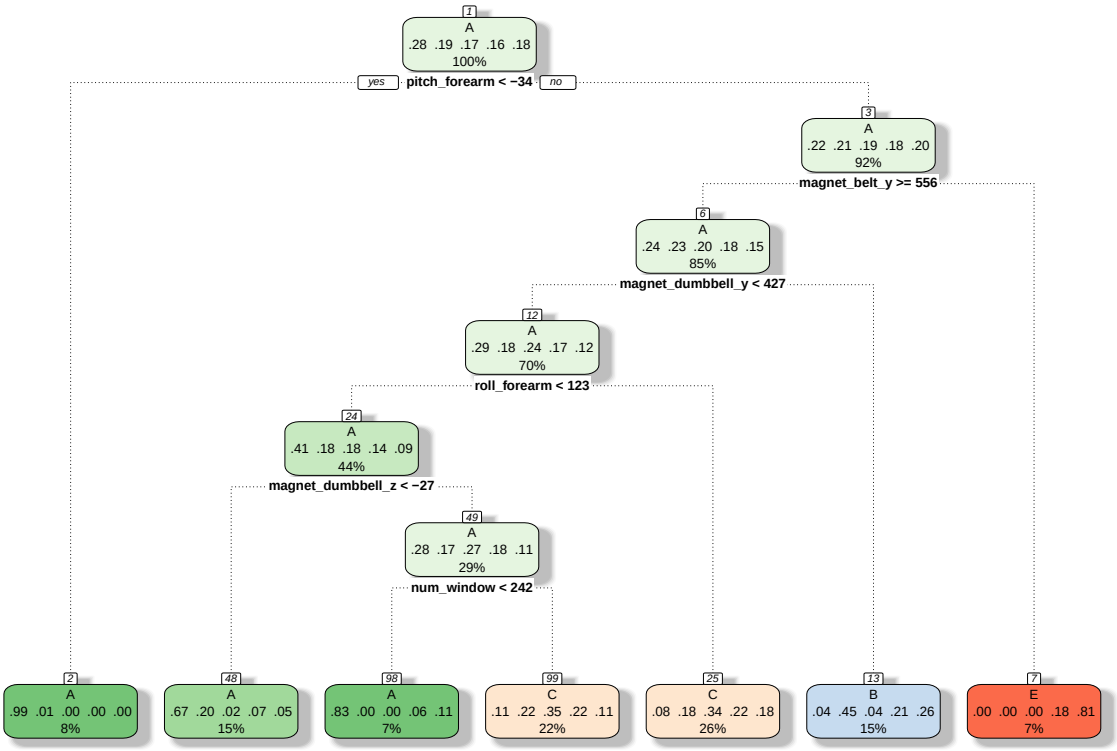
While we're at it - we can do the same for the final testing (i.e. validation) dataset.

```
validate <- read.csv("pml-testing.csv")
validate2 <- validate[-nsv]
validate3 <- validate2 %>%
  select(-c(X, user_name, raw_timestamp_part_1, raw_timestamp_part_2, cvtd_timestamp))
indx <- match(remove, names(validate3))
validate4 <- validate3[,-indx]
validate5 <- validate4[,-redundant]
```

## Generating Models

First we train a decision tree model on the data.

```
modDecTree <- train(classe ~., data = subTrain5, method = "rpart")
fancyRpartPlot(modDecTree$finalModel)
```




Rattle 2018−Oct−20 20:39:16 emilypetrie

The second model we train will be a random forest model.

```
modRandFor <- randomForest(classe ~. , data = subTrain5)
```

## Generating Predictions

Using the models trained above and our testing dataset (not to be confused with the final testing dataset), we can generate predicted values of the movement class variable.

```
predsDecTree <- predict(modDecTree, subTest5)

predsRandFor <- predict(modRandFor, subTest5)
```

## Comparing Model Accuracy

The confusion matrices below indicate that the random forest model does a much better job accurately predicting movement class compared to the decision tree model. This is born out by the calculated sensitivity, specificity, and accuracy measures as well (the decision tree model managed a measly 49.78% accuracy measure while the random forest model predicted with an impressive 99.67% accuracy. This is not surprising, since the random forest method is designed to aggregate the results of multiple decision trees into one prediction.

```
confusionMatrix(predsDecTree, subTest5$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1856  235    9  111  119
##          B   54  550   59  248  270
##          C  322  731 1299  814  547
##          D    0    0    0    0    0
##          E    0    2    1  113  506
##
## Overall Statistics
##
##                Accuracy : 0.5367
##                  95% CI : (0.5256, 0.5478)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.413
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8315   0.3623   0.9496   0.0000  0.35090
## Specificity            0.9156   0.9003   0.6274   1.0000  0.98189
## Pos Pred Value         0.7966   0.4657   0.3499      NaN  0.81350
## Neg Pred Value         0.9318   0.8548   0.9833   0.8361  0.87043
## Prevalence             0.2845   0.1935   0.1744   0.1639  0.18379
## Detection Rate         0.2366   0.0701   0.1656   0.0000  0.06449
## Detection Prevalence   0.2970   0.1505   0.4732   0.0000  0.07928
## Balanced Accuracy      0.8736   0.6313   0.7885   0.5000  0.66639
```

```
confusionMatrix(predsRandFor, subTest5$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    4    0    0    0
##          B    0 1513   10    0    0
##          C    0    1 1358    5    0
##          D    0    0    0 1281    3
##          E    0    0    0    0 1439
##
## Overall Statistics
```

```
##
##                Accuracy : 0.9971
##                  95% CI : (0.9956, 0.9981)
##     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   0.9967   0.9927   0.9961   0.9979
## Specificity            0.9993   0.9984   0.9991   0.9995   1.0000
## Pos Pred Value         0.9982   0.9934   0.9956   0.9977   1.0000
## Neg Pred Value         1.0000   0.9992   0.9985   0.9992   0.9995
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1928   0.1731   0.1633   0.1834
## Detection Prevalence   0.2850   0.1941   0.1738   0.1637   0.1834
## Balanced Accuracy      0.9996   0.9976   0.9959   0.9978   0.9990
```

## Model Application

Since we trained the random forest model on a separate data set from that which it was tested on - it is reasonable to assume that the out-of-sample error is going to approximate the 99.67% accuracy rate just observed.

The final step in our analysis is to run the model selected on the validation dataset.

```
predsRandFor_val <- predict(modRandFor, validate5)
```