Predicting Exercise Performance With the Quantified Self

Rafi Kurlansik

October 20, 2015

Summary The advent of wearable technology has allowed for the collection of human movement data. In an attempt to quantify how well an exercise is performed, a team of researchers used on-body and wearable sensors to record the motion of a specific exercise. These readings were captured using the accelerometer, gyroscope, and magnetometer of sensors places on the arm, forearm, belt, and dumbbell. With the data they collected, we will work to select a machine learning algorithm that will correctly classify new exercise data with the highest accuracy. The original paper can be found here, and the data here.

Exploratory Data Analysis The weight exercise data set is fairly complex, with over 19,000 observations and 160 variables. This structure is too large to display cleanly, so we will merely show a few sample columns. All of the variables are essentially some variation of the ones we are about to see.

```
dim(training)
## [1] 19622   160
dim(testing)
## [1]  20 160
col_samp <- training[1:20, grep("classe|total|user|time|belt_x", colnames(training))]
head(col_samp[, c(1,4,6,7,13)])
##   user_name   cvtd_timestamp var_total_accel_belt gyros_belt_x classe
## 1  carlitos 05/12/2011 11:23                   NA         0.00      A
## 2  carlitos 05/12/2011 11:23                   NA         0.02      A
## 3  carlitos 05/12/2011 11:23                   NA         0.00      A
## 4  carlitos 05/12/2011 11:23                   NA         0.02      A
## 5  carlitos 05/12/2011 11:23                   NA         0.02      A
## 6  carlitos 05/12/2011 11:23                   NA         0.02      A
```

We see the user name, a time stamp for the activity, a sensor summary statistic, a sensor measurement, and the classification `classe`. The classification is a factor variable with 5 levels, each corresponding to a different activity:

Class A - Exactly according to specification (i.e., doing the exercise correctly). Class B - Throwing the Elbow 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.

`classe` is the outcome we will ultimately want our model to predict, but for now there is more exploring to do.

Missing Values When we check for NA's, we see there are quite a lot. Upon further inspection, we see that the testing set is missing all rows in multiple columns. Here is one column, and then the sum of columns with all NA's:

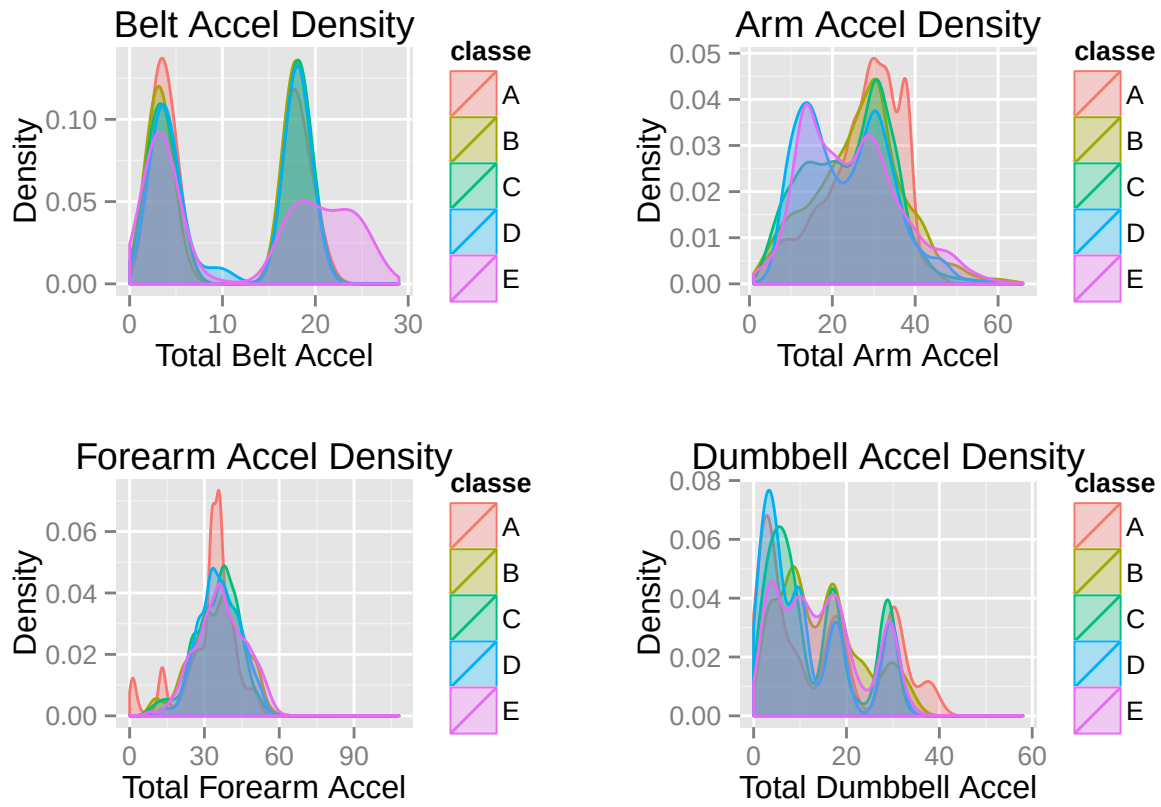
```
sum(colSums(is.na(training)))
## [1] 1287472
sum(colSums(is.na(testing)))
## [1] 2000
```

```
testing$kurtosis_roll_belt
##  [1] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
sum(colSums(is.na(testing)) == nrow(testing))
## [1] 100
```

In fact, 100 of its 160 columns have no data in them! This makes prediction with those variables impossible, so we will drop them now.

```
testing <- testing[, colSums(is.na(testing)) != nrow(testing)]
```

Visualizing Total Acceleration by Class With so many variables and so many possible relationships, visually identifying some patterns is a challenge. One approach is to plot the densities of the `total_accel_group` of variables and get and idea of how acceleration differs at each sensor, for each class.



In the above figure we see several useful features.

*Belt acceleration is highest for the 'E' class, whose description is 'throwing hips to the front'. It also has an otherwise clear bi-modal distribution (you either are really moving your hips or you aren't moving them much at all), with the A class clearly distributed closer to 0.

*Arm acceleration is mostly consistent across different movements, with the exception of lowering the dumbbell only halfway (D) and throwing the hips forward (E).

*Forearm acceleration appears to have relatively similiar distributions across all classes, with less variability in the correct motion.

*Dumbbell acceleration is lower for motions that restrict dumbbell motion (C and D), and higher for full, correct motion (A).

The data reflects the classification variables nicely.

Feature Selection As previously mentioned, 100 of the testing predictors have no values in their rows. To build our training set we can select those columns that match the remaining columns in the test set. We will also remove variables for `X`, which appears to be redundant row names, `user_name`, and the time series variables.

An argument can be made that time series data should be included as each movement is a process with a beginning and end. However, as our density plot above shows, the raw sensor measurements do appear to capture the movements for each class. In the interest of simplicity, we will roll with that.

```
classe <- training$classe   ## Remove classe variable before matching
matches_index <- match(colnames(testing), colnames(training))
training <- training[, matches_index[-60]]  ## Drop problem_id column which became NA
training$classe <- classe   ## Put back classe variable
training <- training[-c(1,2,3,4,5,6,7)]     ## Remove username, X, and time series variables
```

As a final test, we can call the `nearZeroVar` function to see if any of our remaining predictors have little variability, and thus little information to contribute to the model.

```
nsv <- nearZeroVar(training, saveMetrics = T)
unique(nsv$nzv)
## [1] FALSE
```

No near zero variance predictors. Good!

Model Selection Moving forward, we have narrowed down our predictors from 159 to 53. We will use randomForest as our model. Random forest is a good choice for this data set because we are mostly interested in prediction, not interpretability. Cross validation is achieved within the random forest algorithm itself.

Each tree is built using a random sampling of our original predictors in the training set. Some of the rows for predictors of the tree are held out, and this group is called the out-of-bag sample. When the tree is fully grown and there are no more splits to be made, the OOB samples are run through the tree. How accurately the tree predicts these OOB samples yields an error rate. The OOB error rate reported in the final model is the rate for how all trees, in aggregate, predicted the OOB samples. *This is akin to separating data into train and test sets to estimate the out-of-sample error rate.*

We will use the OOB error rate on our training set as our estimate for the out-of-sample error, and the final error rate will be our true out-of-sample error.

Now construct the model, and evaluate:

```
set.seed(206)
fit.rf <- randomForest(classe ~ ., data = training, importance = TRUE)
fit.rf
##
## Call:
##  randomForest(formula = classe ~ ., data = training, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.28%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 5577    2    0    0    1 0.0005376344
## B   11 3784    2    0    0 0.0034237556
## C    0    9 3410    3    0 0.0035067212
## D    0    0   18 3196    2 0.0062189055
## E    0    0    1    5 3601 0.0016634322
```

Accuracy looks promising, with an OOB estimate of error rate of 0.28%!

Before moving to prediction,

-summary -EDA -feature selection -model selection/training set evaluation -prediction -evaluating results -conclusions