Exercise Quality Prediction

Drew Herring

September 10, 2017

Background

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, we will attempt to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants to predict the fashion in which the exercise was completed. 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 (http://groupware.les.inf.puc-rio.br/har) (see the section on the Weight Lifting Exercise Dataset).

Getting, Cleaning and Exploratoration

First, we set the seed to ensure reproducibility and then download the datasets. Libraries we will be using are: dplyr, caret, rpart, and randomForest.

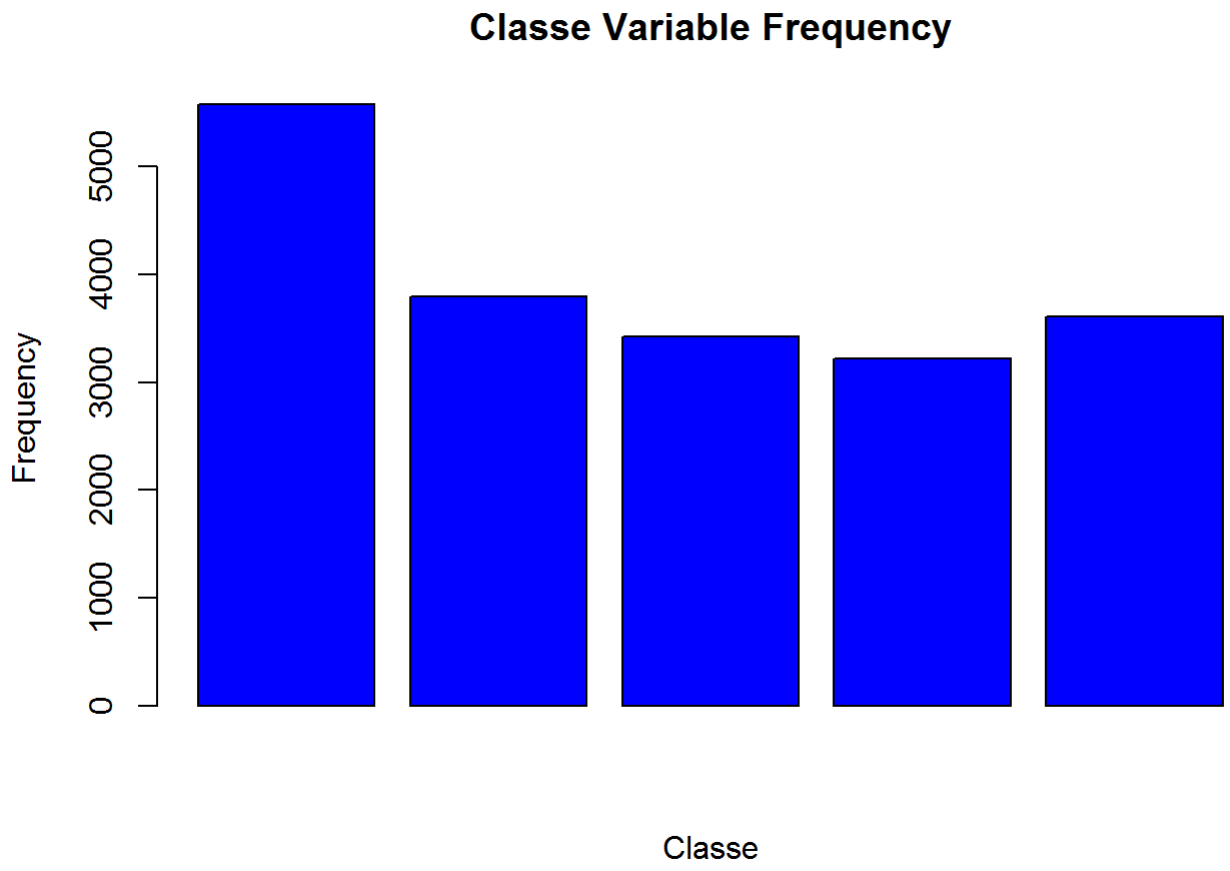
```
set.seed(3)

#download data
if(!file.exists('pml-training.csv')){
  download.file('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv', destfile
 = 'pml-training.csv')
}
if(!file.exists('pml-testing.csv')){
  download.file('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv', destfile
= 'pml-testing.csv')
}

#Load data
train_data = read.csv('pml-training.csv')
test_data = read.csv('pml-testing.csv')

#expolre data a bit
#str(train_data)
```

After using the str() function on the training data, we see it contains 19,622 rows and 160 columns. The variable we would like to predict is ‘classe’, and the figure below show the different catagories with their distributions in the training dataset.



There are also many ‘NA’ values within the dataset, and before we can use the data to build a prediction model we need to tidy up a bit.

The first few columns will also not make good predictors as they only help to describe the other variables. For example, they contain timestamp, user_name, etc.

```
test_data %>% select_if(colSums(is.na(train_data)) == 0) -> test_data
train_data %>% select_if(colSums(is.na(train_data)) == 0) -> train_data

train_data %>%
  select(-c(X,user_name,raw_timestamp_part_1,raw_timestamp_part_2,cvtd_timestamp,new_window)) ->
 train_data
test_data %>%
  select(-c(X,user_name,raw_timestamp_part_1,raw_timestamp_part_2,cvtd_timestamp,new_window)) ->
 test_data
```

Modeling

Before we begin building the model, it is important to split our training data into two sets in order to perform cross validation. For this exercise, we will be splitting the data 60/40, with 40% being used for evaluation of the model only. The training test partition will not be used to develop the model, and will only be for accessing accuracy of our model.

```
train_par <- createDataPartition(y = train_data$classe, p = 0.60, list=FALSE)
train_train <- train_data[train_par,]
train_test <- train_data[-train_par,]
```

The new partition of training data resulted in 11,776 records for training, and 7,846 for testing.

We want to also remove columns containing few unique values, as this could confound our model. For this we will be using the nearZeroVar function within the caret package.

```
nzv_col <- nearZeroVar(train_train)
train_train <- train_train[ , -nzv_col]
train_test <- train_test[ , -nzv_col]
test_data <- test_data[ , -nzv_col]
```

After removing variables with 'NA' values and variables with low variance, we are left with 53 remaining variables. Since the dataset is relatively small, we will attempt to build a model using all 53 variables and ensure we have an acceptable accuracy.

Decision Tree Model

Using our training partition data, we will attempt to build a decision tree model using all 53 variables, make predictions and check the accuracy.

```
#build a decision tree model
dt_pred_model <- rpart(classe ~ ., data = train_train, method = 'class')

#make predictions with our testing subset of training data
dt_pred_val <- predict(dt_pred_model, train_test, type = 'class')
#check the accuracy of the model
dt_conf_mat <- confusionMatrix(dt_pred_val, train_test$classe)
```

According to the confusion matrix (figure 1 in the appendix), the accuracy of the decision tree model is 75.8% with an out of sample error rate of 24.2%.

Random Forest Model

Reusing our training partition data, we will attempt to build a random forest model using all 53 variables, make predictions and check the accuracy.

```
#build a random forest model
rf_pred_model <- randomForest(classe ~ ., data = train_train, ntree = 100, importance = TRUE)

#make predictions with our testing subset of training data
rf_pred_val <- predict(rf_pred_model, newdata = train_test)
#check the accuracy of the model
rf_conf_mat <- confusionMatrix(rf_pred_val, train_test$classe)
```

According to the confusion matrix (figure 2 in the appendix), the accuracy of the random forest model is 99.5% with an out of sample error rate of 0.5%.

Conclusion

To predict on the classe variable we chose the random forest model as the accuracy is much higher than the decision tree model. We can now use the actual test data and predict the fashion in which particpants completed the exercise.

Appendix

Figure 1: Decision Tree Confusion Matrix

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1872  199   23   53   44
##          B  163 1002  143  126   63
##          C   11   82 1024   67   54
##          D  119  159  147  845   77
##          E   67   76   31  195 1204
##
## Overall Statistics
##
##                Accuracy : 0.758
##                  95% CI : (0.7483, 0.7674)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6941
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8387   0.6601   0.7485   0.6571   0.8350
## Specificity            0.9432   0.9218   0.9670   0.9235   0.9424
## Pos Pred Value         0.8544   0.6693   0.8271   0.6273   0.7654
## Neg Pred Value         0.9363   0.9187   0.9479   0.9321   0.9621
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2386   0.1277   0.1305   0.1077   0.1535
## Detection Prevalence   0.2793   0.1908   0.1578   0.1717   0.2005
## Balanced Accuracy      0.8909   0.7909   0.8578   0.7903   0.8887
```

Figure 2: Random Forest Confusion Matrix

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    7    0    0    0
##          B    0 1505    8    0    0
##          C    0    6 1360   11    0
##          D    0    0    0 1275    8
##          E    0    0    0    0 1434
##
## Overall Statistics
##
##                Accuracy : 0.9949
##                  95% CI : (0.9931, 0.9964)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9936
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9914   0.9942   0.9914   0.9945
## Specificity            0.9988   0.9987   0.9974   0.9988   1.0000
## Pos Pred Value         0.9969   0.9947   0.9877   0.9938   1.0000
## Neg Pred Value         1.0000   0.9979   0.9988   0.9983   0.9988
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1918   0.1733   0.1625   0.1828
## Detection Prevalence   0.2854   0.1928   0.1755   0.1635   0.1828
## Balanced Accuracy      0.9994   0.9951   0.9958   0.9951   0.9972
```