Exercise Prediction Project

Billy Caughey

2018-03-18

Introduction

Tree-based methods can be powerful tools in prediction. In the current project, Tree-based methods will be used to predict exercise movements of 6 study participants. Information about these exercises where collected by accelerometers were worn by study participants around the belt, forearm, and arm. As part of the study, participants were asked to perform an exercise correctly and incorrectly.

This project will have three main components: 1) training a model, 2) testing the model, and 3) making predictions with the model.

Training the Model

Observing the Training Data

Training and testing data were supplied for this project. The training data contains observations with identified exercise movements. Alternatively, the 'testing', or what will be referenced as the predicting data, has 20 observations with no classification of exercise.

This, initially, causes some concern. If the training data is not sufficiently large to train and test, then additional measures will be taken to correct this before any predictions can occur. Now, consider the size of the training data.

```
trainData <- readRDS(paste0(personal,
                            "Coursera/Machine Learning/Final Project/mlfinal_trainingdata.rds"))
nrow(trainData)
```

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

With 19,622 observations, there is more than enough information to split this set into a training (labeled as t.data) and testing set (labeled as v.data). Using a 70/30 split, this is done below. It should be noted this new testing set will be where the testing error, or out of sample, will be computed. For the entirety of this project, the seed of 1234 will be used.

```
set.seed(1234)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
inTrain <- createDataPartition(trainData$classe, p = 0.7, list = FALSE)
t.data <- trainData[inTrain,]
v.data <- trainData[-inTrain,]
dim(t.data)
```

```
## [1] 13737   160
```

Data Cleaning

Before the model can be trained, the data used to train will be cleaned and features created if needed. It should be noted and emphasized any cleaning which occurs in the training data will occur in both the test and prediction sets.

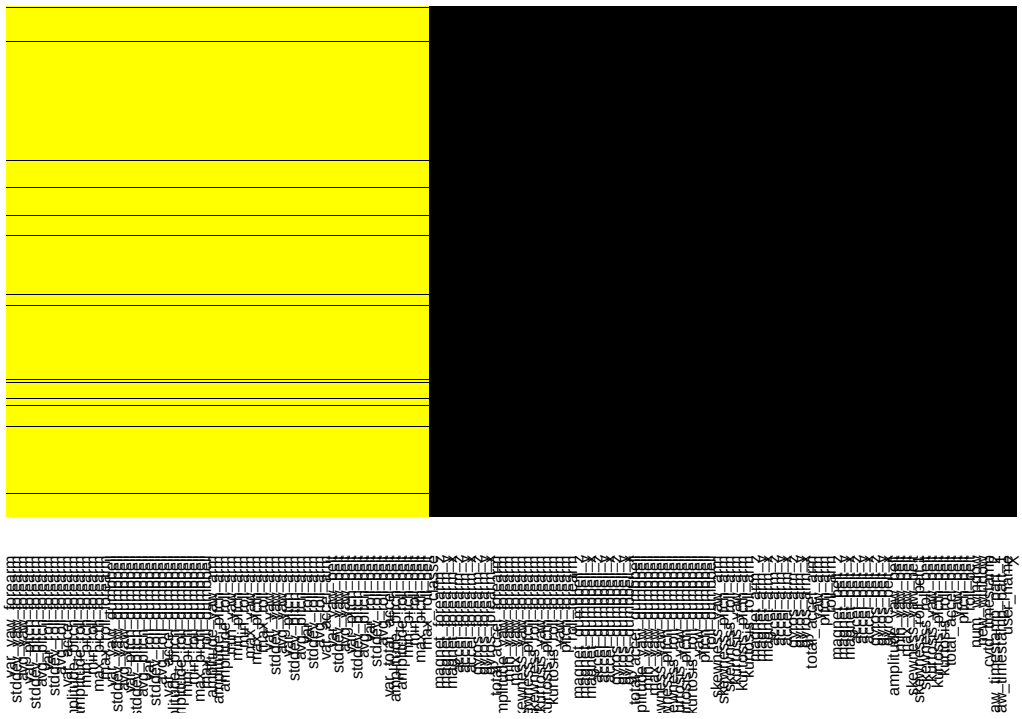
A first question is does the fields in the training data contain any missing data? To answer this question, a missingness map is used from the Amelia library. This map will display where missing data exists.

```
library(Amelia)
```

```
## Loading required package: Rcpp
```

```
## ##
## ## Amelia II: Multiple Imputation
## ## (Version 1.7.4, built: 2015-12-05)
## ## Copyright (C) 2005-2018 James Honaker, Gary King and Matthew Blackwell
## ## Refer to http://gking.harvard.edu/amelia/ for more information
## ##
```

```
missmap(t.data, col = c("yellow","black"), title = "Missingness Map",y.at=c(1),
        y.labels=c(''),legend = FALSE,x.cex = 0.5)
```

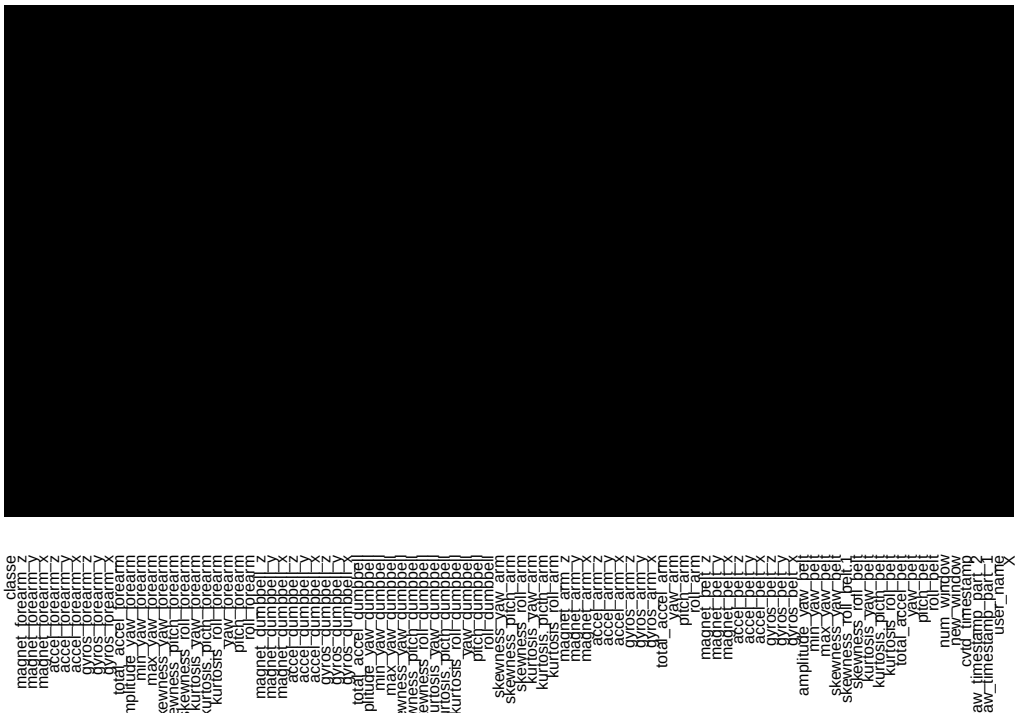


In the missingness map, the yellow represents all the missing dat observed in the training set. As this missingmess plot suggests, there is a multitude of fields which are empty. These fields are removed from the analysis.

```
t.data <- t.data[lapply(t.data, function(x) sum(is.na(x))/length(x)) < 0.1]
missmap(t.data, col = c("yellow","black"), title = "Missingness Map",y.at=c(1),
```

```
        y.labels=c(''),legend = FALSE,x.cex = 0.5)
```

Missingness Map




```
dim(t.data)
```

```
## [1] 13737    93
```

A second question is are there are any significantly sparse fields in the training set? The answer to this question is yes. There are several fields which contain less than sufficient information to impute missing values. Therefore, these fields will be dropped.

```
dropFields <- grep("timestamp|X|user_name|window|skewness|kurtosis|min|max|amplitude",
                   names(t.data))
t.data <- t.data[,-dropFields]
names(t.data)
```

```
##  [1] "roll_belt"            "pitch_belt"           "yaw_belt"
##  [4] "total_accel_belt"     "gyros_belt_x"         "gyros_belt_y"
##  [7] "gyros_belt_z"         "accel_belt_x"         "accel_belt_y"
## [10] "accel_belt_z"         "magnet_belt_x"        "magnet_belt_y"
## [13] "magnet_belt_z"        "roll_arm"             "pitch_arm"
## [16] "yaw_arm"              "total_accel_arm"      "gyros_arm_x"
## [19] "gyros_arm_y"          "gyros_arm_z"          "accel_arm_x"
## [22] "accel_arm_y"          "accel_arm_z"          "magnet_arm_x"
## [25] "magnet_arm_y"         "magnet_arm_z"         "roll_dumbbell"
## [28] "pitch_dumbbell"       "yaw_dumbbell"         "total_accel_dumbbell"
## [31] "gyros_dumbbell_x"     "gyros_dumbbell_y"     "gyros_dumbbell_z"
## [34] "accel_dumbbell_x"     "accel_dumbbell_y"     "accel_dumbbell_z"
## [37] "magnet_dumbbell_x"    "magnet_dumbbell_y"    "magnet_dumbbell_z"
```

```
## [40] "roll_forearm"         "pitch_forearm"        "yaw_forearm"
## [43] "total_accel_forearm"  "gyros_forearm_x"      "gyros_forearm_y"
## [46] "gyros_forearm_z"      "accel_forearm_x"      "accel_forearm_y"
## [49] "accel_forearm_z"      "magnet_forearm_x"     "magnet_forearm_y"
## [52] "magnet_forearm_z"     "classe"
```

```
dim(t.data)
```

```
## [1] 13737    53
```

A third question would be is there a need for feature engineering? Any adjustments to the data by scale or transform would not affect the model. Therefore, there is not significant need for feature engineering.

Training

With the cleaning complete, a model now can be trained. A 5-fold cross validation will be applied to a random forest. The results of training the model are below.

```
set.seed(1234)
train_control <- trainControl(method="cv",number=5)
mod1 <- train(classe ~ . , data = t.data, trControl = train_control, method = "rf")
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

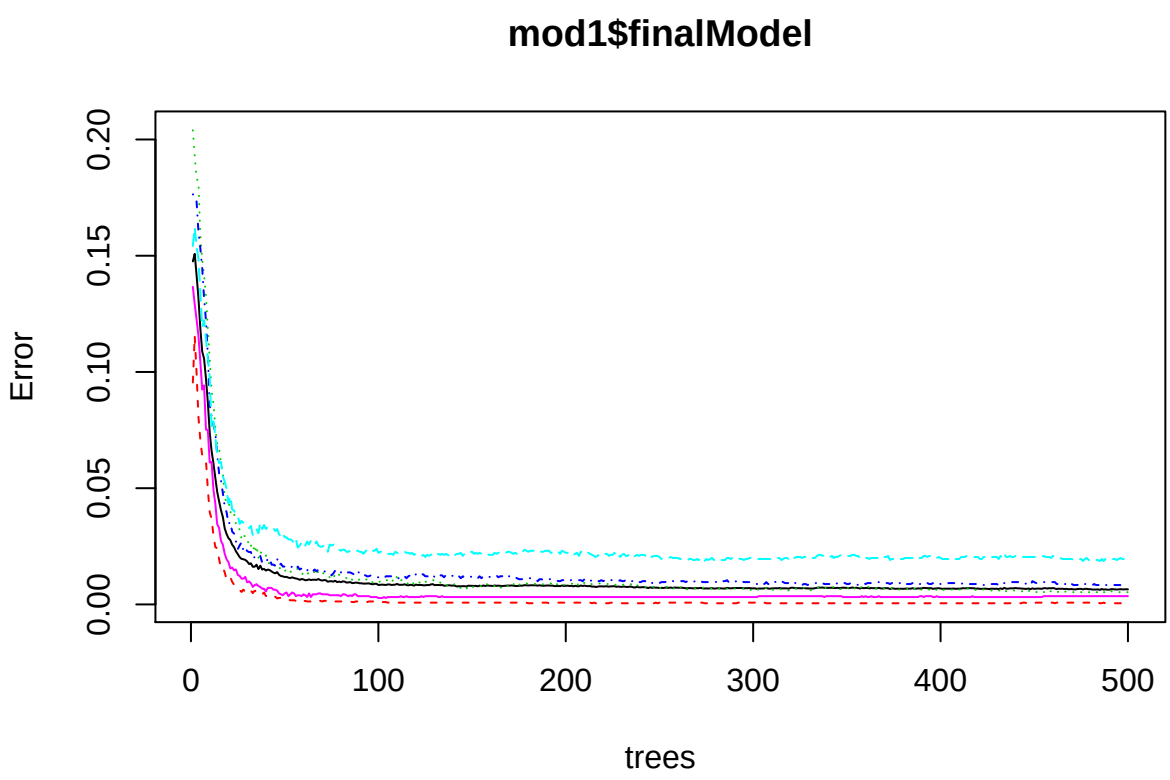
```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
print(mod1)
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10988, 10990, 10989, 10991, 10990
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9909728  0.9885792
##   27    0.9906092  0.9881191
##   52    0.9864596  0.9828693
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```
plot(mod1$finalModel)
```



The training error associated with this model is 0.9% (1 - accuracy). This is a great error rate, but must be observed with some caution. Typically, the training error is an overestimataion of the testing error rate.

A visual representation of the model was desired at this point. Unfortunately, random forests using the 'train' function do not lend themselves to such visual representations.

Testing the Model

Using the testing set previously established, the model will be tested. From this test, the testing error will be established. This testing error will be used to determine the appropriateness of the model.

```
v.data <- v.data[,lapply(v.data, function(x) sum(is.na(x))/length(x)) < 0.1]
dropFields <- grep("timestamp|X|user_name|window|skewness|kurtosis|min|max|amplitude",
                   names(v.data))
v.data <- v.data[,-dropFields]

valid.model <- predict(mod1,v.data)

table(valid.model,v.data$classe)
```

```
##
## valid.model    A    B    C    D    E
##           A 1674    8    0    0    0
##           B    0 1130    7    0    0
##           C    0    1 1018   10    1
##           D    0    0    1  954    1
```

```
##            E    0    0    0    0 1080
```

```
mean.class.error <- mean((valid.model != v.data$classe)); print(mean.class.error)
```

```
## [1] 0.004927782
```

```
accuracy <- 1 - mean((valid.model != v.data$classe)); print(accuracy)
```

```
## [1] 0.9950722
```

The resulting testing error is 0.5%. This is a phenomenal testing error. There is one point of concern. The testing error is less than the training error. Typically, the training error is less than testing error due to training error being an overestimation of the testing error. With that said, it is not impossible for the testing error to be less than the training error. In this case, since the training and testing error are essentially the same. If the testing error was significantly higher than the training error there would be cause for concern.

Prediction

The conclusion of this project is to predict which exercise twenty subjects were performing. The testing data will be cleaned in the same fashion as the training and validation data were.

```
testData <- readRDS(paste0(personal,
               "Coursera/Machine Learning/Final Project/mlfinal_testdata.rds"))

## Need to clean the test data the same way training and validation were cleaned
testData <- testData[,lapply(testData, function(x) sum(is.na(x))/length(x)) < 0.1]
dropFields <- grep("timestamp|X|user_name|window|skewness|kurtosis|min|max|amplitude",
                   names(testData))
testData <- testData[,-dropFields]

test.predict <- predict(mod1,testData)
test.predict
```

```
##  [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
```