Exercise Prediction with Random Forests

Christian Andrews

November 10, 2017

Executive summary

Using activity trackers 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, or to find patterns in their behavior. In this project, the 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: 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).

(Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E).

Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes

Read more: http://groupware.les.inf.puc-rio.br/har#ixzz4y5Bm6Y2o (http://groupware.les.inf.puc-rio.br/har#ixzz4y5Bm6Y2o)

The goal of your project is to predict the manner in which they did the exercise. This is the “classe” variable in the training set.

The final resulting model performed very well with a 99.94% accuracy and did not appear to suffer significant overfitting.

Data import and clean up:

The training data for this project are available here (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv).

The test data are available here (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv).

The Caret package is required for this analysis.

```
library(caret)
set.seed(314159)
```

Import the data sets and capture NA and Divide by 0 errors

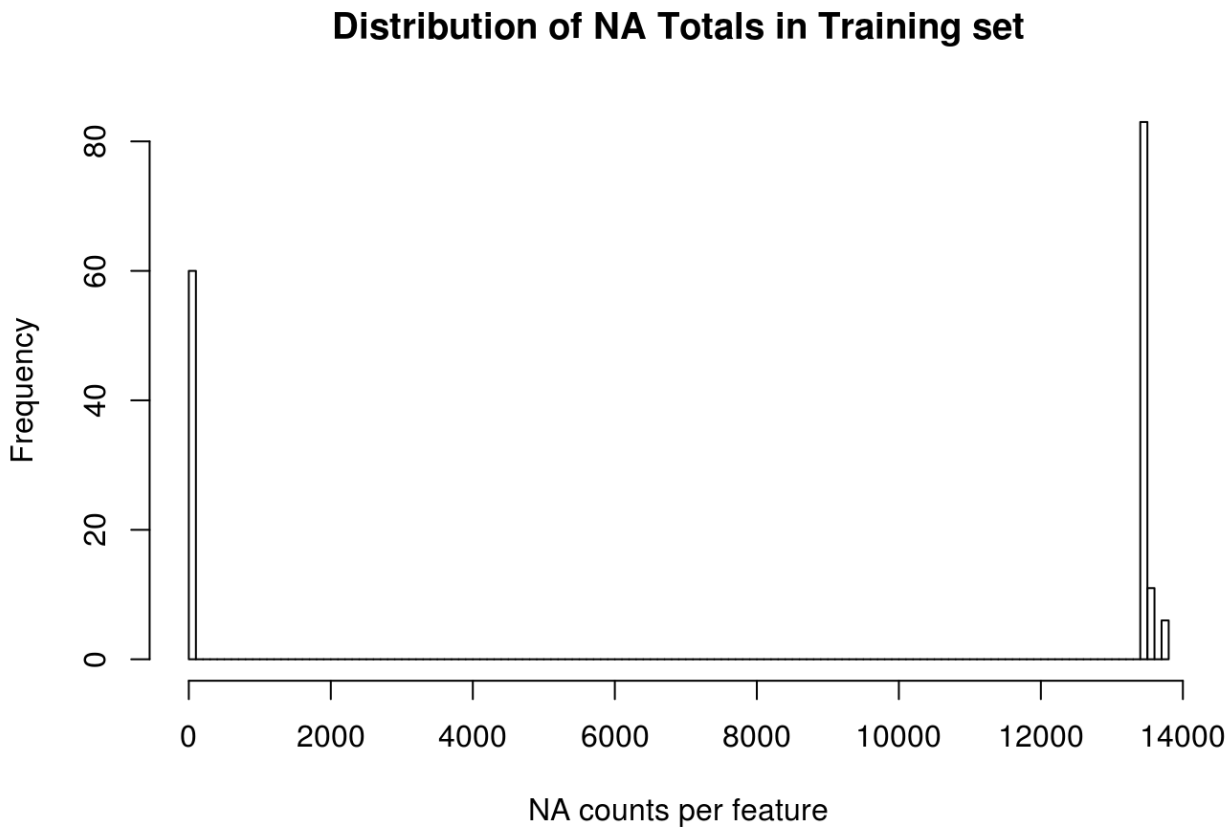
```
trainingSetFull <- read.csv("./data/pml-training.csv",na.strings=c("NA","","#DIV/0!"))
testingSet <- read.csv("./data/pml-training.csv",na.strings=c("NA","","#DIV/0!"))
```

Begin by creating a training and validation set. Then calculate the nymber of features that have a significant amount of missing infomation.

```
#Built training and validation sets: 70% / 30%
inTrain <- createDataPartition(y = trainingSetFull$classe, p = 7/10, list = FALSE)
trainingSet <- trainingSetFull[inTrain, ]
validationSet <- trainingSetFull[-inTrain,]

#Calculate NA total by feature
cummNaInColumn <- apply(trainingSet, 2, function(d){sum(is.na(d))})

hist(cummNaInColumn, breaks=100, xlab='NA counts per feature', main='Distribution of NA Totals in
Training set')
```



Judging by the results of the histogram, there are many features with very little information. These features will be removed along with columns that are not relevant to the exercise (pun intended).

User name, New Window and time stamp variables should be ignored since they are not relevant to building a generalized model to predict the activity based on tracker data.

```
#Remove features:
cummNaInColumn[1]=1000 #user Name
cummNaInColumn[2]=1000 #raw time stamp 1
cummNaInColumn[3]=1000 #raw time stamp 2
cummNaInColumn[4]=1000 #cvtd time stamp
cummNaInColumn[5]=1000 #new window
cummNaInColumn[6]=1000 #new window int

#Only take features with a full dataset (where NA count is 0)
cleantrainingSet <- trainingSet[, which(cummNaInColumn == 0)]
```

The Model (and cross-validation):

Typically, for a classification problem with p features, √p (rounded down) features are used in each split. In this case we have 51 features, so we will use floor(√51) = 7 folds for cross-validation. Fewer folds will yeild results faster but precision may suffer. Additional folds will take a significant amount of time to calculate (trust me).

In this case, I chose to use the random forest model.

```
forestFit <- train(classe ~ ., method = "rf", data = cleantrainingSet, trControl = trainControl(m
ethod = "cv", number = 7))
```

Model Results

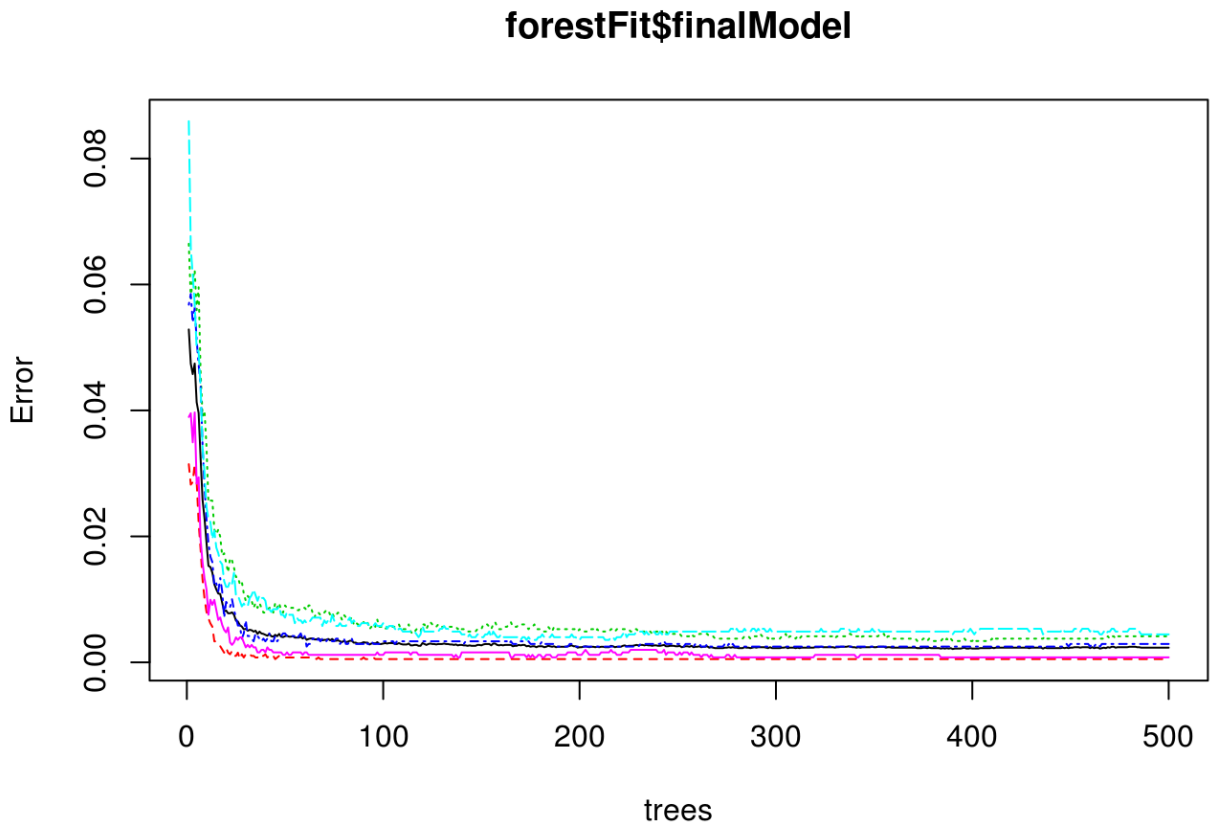
Reviewing the model, specially the Confusion Matrix below, you can see that each excerise type A to E had a very low classificaiton rate rate. This is good but may present a problem with overfitting to the test data to follow.

```
print(forestFit$finalModel)
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 0.23%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3904    1    0    0    1 0.0005120328
## B    7 2647    3    1    0 0.0041384500
## C    0    6 2389    1    0 0.0029215359
## D    0    0   10 2242    0 0.0044404973
## E    0    0    0    2 2523 0.0007920792
```

Additional information about the model can be seen by the plot of the final model.

```
plot(forestFit$finalModel)
```



Model Validation

Now to test the Random Forest model against the validation data. This is done to pre-test before testing on the actual, final, real test data to see the out of sample error rate we should expect. From the information below you can see we should expect a 99.8% Accuracy or 0.02% Out of Sample Error.

```
predRF <- predict(forestFit, validationSet)

confusionMatrix(predRF, validationSet$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    2    0    0    0
##          B    0 1137    1    0    2
##          C    0    0 1025    5    0
##          D    0    0    0  958    1
##          E    0    0    0    1 1079
##
## Overall Statistics
##
##                Accuracy : 0.998
##                  95% CI : (0.9964, 0.9989)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9974
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9982   0.9990   0.9938   0.9972
## Specificity            0.9995   0.9994   0.9990   0.9998   0.9998
## Pos Pred Value         0.9988   0.9974   0.9951   0.9990   0.9991
## Neg Pred Value         1.0000   0.9996   0.9998   0.9988   0.9994
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1932   0.1742   0.1628   0.1833
## Detection Prevalence   0.2848   0.1937   0.1750   0.1630   0.1835
## Balanced Accuracy      0.9998   0.9988   0.9990   0.9968   0.9985
```

Final Testing

The model perfomed very well with an Accuracy of 99.94% but slightly higher than expected 0.06% Out of Sample Error.

```
predFinalTest <- predict(forestFit, newdata = testingSet)
accuracy <- sum(predFinalTest == testingSet$classe)/length(predFinalTest)
oos_error <- 1 - accuracy
print(oos_error)
```

```
## [1] 0.0006115585
```