But How Well Are You Doing It? Predicting from Fitbit Data

Travis Fell

November 15, 2015

I. Introduction

This analysis predicts how well test subjects would perform a dumbbell bicep curl from personal movement tracking devices like Nike FuelBand and Fitbits.

Using a training set of measurements from these personal exercise movement devices like Nike FuelBands and FitBits as well as assessments from professional trainers on the correctness of exercises performed, this analysis will predict how well 20 test subjects performed the same exercise. Exercise performance is categorized by the classe column in the training set using the following levels: - A: Exactly according to the specification - B: Throwing the elbows to the front - C: Lifting the dumbbell only halfway - D: Lowering the dumbbell only halfway - E: Throwing the hips to the front Read more here: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)

II. Set Up Work

First, we'll start out by loading libraries, loading data and doing a bit of exploratory analysis.

```
setwd("C:/Users/fellt/Desktop/Data Science/Coursera Data Science Specialization/08 - Prac
tical Machine Learning/8-PracticalMachineLearning")
set.seed(333)
library(caret)
library(randomForest)
library(ggplot2)
library(dplyr)
library(parallel, quietly=T)
library(doParallel, quietly=T)
train <- read.csv("pml-training.csv", stringsAsFactors = FALSE, na.strings=c("", " ", "N
A"))
test <- read.csv("pml-testing.csv", stringsAsFactors = FALSE, na.strings=c("", " ", "N
A"))
summary(train)
str(train)
View(train)
```

This data set has a lot of extraneous data. The first 7 columns have no predictive value with respect to performing an exercise. Furthermore, there are a LOT of columns with little to no data.

III. Preparing the Data Set

We need tidy up this this data by squeezing out columns that are not useful for predicting and removing rows with missing data.

```
#remove first 7 columns from both data sets as they are only informational about the subj
ects and test
train <- train[,8:160]
test <- test[,8:160]

# filter out columns with majority NA values
train <- train[sapply(train, function(x) !any(is.na(x)))]
test <- test[sapply(test, function(x) !any(is.na(x)))]

# find columns with more than 15% of rows = 0 and remove
  zerotest <<- NULL
  for(i in 1:length(train))
    {
    zerotest <<- c(zerotest, nrow(train[train[,i] == 0,])/nrow(train))
  }
removecols <- colnames(train[,which(zerotest >.15)])
train[,removecols] <- list(NULL)
test[,removecols] <- list(NULL)
```

In addition to removing columns with missing data, we are also going to find pairs of columns that correlate with each other and remove one from the data set. Then, we'll drop records missing data.

```
#find correlations in remaining features to ID add'l variables to exclude
traincor <- findCorrelation(cor(train[,1:length(train)-1]), cutof = .8, verbose = FALSE)
#leave off dependent variable for this command
train <- train[,-traincor] #exclude 1 feature from each pair of highly correlated feature
s from training set
test <- test[, -traincor] #exclude same features from test set

# find records with any 0 values and remove
train[train == 0] <- NA
train <- train[complete.cases(train),]
train$classe <- as.factor(train$classe)
```

The processing above yields 33 columns with predictive value in both the training and test set.

III. Analyze the Data

Now that we have a nice and tight data set, we'll perform some cross-validation on the training set which will enable us to to estimate out of sample error on our models.

```
# split the training set into 70/30 training-validation sets
inTrain <- createDataPartition(train$classe, p = .7, list = FALSE)
trainMod <- train[inTrain,]
validMod <- train[-inTrain,]
```

Now, for the fun stuff. We will use the updated training set, trainMod, to build some models and the validation set, validMod, to estimate the out of sample error. We'll start with a generalized boosting model.

```
# turn on parallel processing to help improve performance
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)

#create boosting model
load("boostingMod.rda") #to enable knitr to run quickly, will simply load up saved GBM mo
del
#trainGBM <- train(classe ~ ., method = "gbm", data = trainMod, verbose = FALSE) #note: c
ommented out this line to enable knitr to run in a timely manner
#save(trainGBM, file = "boostingMod.rda") #since this takes a while to run, let's save th
e model to quickly load and reuse later
print(trainGBM)
```

```
#perform cross validation and estimate OOS error on boosting model
validationPredGBM <- predict(trainGBM, validMod)
confusionMatrix(validMod$classe, validationPredGBM)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1016    8    8    7    0
##          B   32  844   17    3    1
##          C    3   32  714    5    4
##          D    8    3   35  668   10
##          E    3    4   11    6  845
##
## Overall Statistics
##
##                Accuracy : 0.9533
##                  95% CI : (0.9466, 0.9595)
##     No Information Rate : 0.2477
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9414
##  Mcnemar's Test P-Value : 7.778e-08
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9567   0.9473   0.9096   0.9695   0.9826
## Specificity            0.9929   0.9844   0.9874   0.9844   0.9930
## Pos Pred Value         0.9779   0.9409   0.9420   0.9227   0.9724
## Neg Pred Value         0.9858   0.9861   0.9799   0.9941   0.9956
## Prevalence             0.2477   0.2078   0.1831   0.1607   0.2006
## Detection Rate         0.2370   0.1969   0.1666   0.1558   0.1971
## Detection Prevalence   0.2424   0.2092   0.1768   0.1689   0.2027
## Balanced Accuracy      0.9748   0.9658   0.9485   0.9770   0.9878
```

The GBM model yields a 95% accuracy rate when ran against the cross-validation set. Not bad, but can we do better with a random forest model?

```
#create random forest model
trainRF <- randomForest(classe~., trainMod)
save(trainRF, file = "rfMod.rda")
print(trainRF)
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = trainMod)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 1.44%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 2416    4    1    4    0 0.003711340
## B   23 2052   19    0    2 0.020992366
## C    4   29 1736    1    1 0.019762846
## D    7    1   43 1639    1 0.030751035
## E    1    1    2    0 2026 0.001970443
```

```
#perform cross validation and estimate OOS error on random forest model
validationPredRF <- predict(trainRF, validMod)
confusionMatrix(validMod$classe, validationPredRF)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1037    0    1    1    0
##          B   16  871   10    0    0
##          C    0    8  750    0    0
##          D    5    0   16  702    1
##          E    0    0    4    3  862
## 
## Overall Statistics
##                                           
##                Accuracy : 0.9848          
##                  95% CI : (0.9807, 0.9883)
##     No Information Rate : 0.2468          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.981           
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9802   0.9909   0.9603   0.9943   0.9988
## Specificity            0.9994   0.9924   0.9977   0.9939   0.9980
## Pos Pred Value         0.9981   0.9710   0.9894   0.9696   0.9919
## Neg Pred Value         0.9935   0.9976   0.9912   0.9989   0.9997
## Prevalence             0.2468   0.2050   0.1822   0.1647   0.2013
## Detection Rate         0.2419   0.2032   0.1749   0.1638   0.2011
## Detection Prevalence   0.2424   0.2092   0.1768   0.1689   0.2027
## Balanced Accuracy      0.9898   0.9916   0.9790   0.9941   0.9984
```

Bam! The Random Forest model gives us an OOS error rate of 1.46% and an accuracy rate of 98%. Let’s use this model to run against the test set and produce files for the final assignment submission.

IV. Predict on the Test Set and Submit

Here we’ll put the test set through the RF prediction model above and create the files for the assignment submission.

```
#make predictions on test data, save to file for submission
testPredRF <- predict(trainRF, test)

# create answers vector with values from test set prediction
testPredRF <- as.character(testPredRF)

# write to file
pml_write_files = function(x){
  n = length(x)
  for(i in 1:n){
    filename = paste0("problem_id_",i,".txt")
    write.table(x[i],file=filename,quote=FALSE,row.names=FALSE,col.names=FALSE)
  }
}
pml_write_files(testPredRF)
```