Predicting Workout Form Using Machine Learning

Christopher Campbell

March 23, 2017

```r
knitr::opts_chunk$set(echo = TRUE)
library(ggplot2)
library(dplyr)
library(caret)
library(Hmisc)
library(GGally)
library(e1071)
```

```r
load("variables.RData")

my_url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
my_url2 <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

download.file(my_url, "training")
download.file(my_url2, "testing")


testing <- read.csv("testing")
training <- read.csv("training")

## I subsequently move testing and training into the data folder above;
## this is a simple way to organize my git.ignore file
```

Introduction

In this paper we analyse the Weight Lifting Exercise Dataset (WLE) from Groupware-LES. This data was collected by taking measurements of weight lifters doing an exercise in 5 different ways (the variable "classe" encodes these). The measurements were obtained using gyroscopes on the arm, the forearm, the waist, and a dumbbell.

Data Preparation

We first prepare the data for analysis by creating a training, a test, and a validation set. The data is of medium size with ~20,000 observations. However, the goal of this analysis is to produce a good estimate of the out-of-sample error rate for 20 samples, per the course requirements of Johns Hopkins University's Intro to Machine Learning. Therefore, while we treat our test set produced in this analysis as truly an untouched sample of the data, we won't be excessively concerned about our algorithm's performance on the test set. The implication, then, is that we won't need a huge test set - we break it up here into 80% training (further broken into 60% training and 20% validation), and 20% testing.

We will create 5 of said data partitions among the training set so that we may do 5-fold cross-validation among our chosen algorithm. Note: we make use of the Caret package (which does do a hyperpararmeter search), but after our model selection, we still will need a good prediction of our out-of-sample error rate.

```r
set.seed(1233)
fortest <- createDataPartition(training$classe,p=.2,list=FALSE)
test <- training[fortest,]

## We need to loop this upcoming process over and over to get an estimate on our out-of-sample
## error (referring to the test set)

pretrain <- training[-fortest,]

forvalidation <- createDataPartition(pretrain$classe,p=.25,list=FALSE) #.25*.8=.2 obvi
train <- pretrain[-forvalidation,]
validation <- pretrain[forvalidation,]
```

Exploratory Data Analysis

We explore the pretrain object, which is our undifferentiated training set. The data has 15,695 observations with 160 variables. From the user_name field, we see that 6 individuals participated in producing the data. Thus, 6 individuals repeated 5 different forms of each exercise. Some information is provided which we will not be concerned about including time_stamps and "window" variables, which have little variation.

One immediate observation is that the variables for each of the bands include a decomposition of role, pitch, yaw, and acceleration many other components including directional information and descriptive statistics about that information. Most of the descriptive statistics are not tabulated for each observation so these can be automatically thrown away (in this case, the mean, standard deviation, kurtosis, and skewness are not sufficient statistics for our class probabilities in the classification models below).

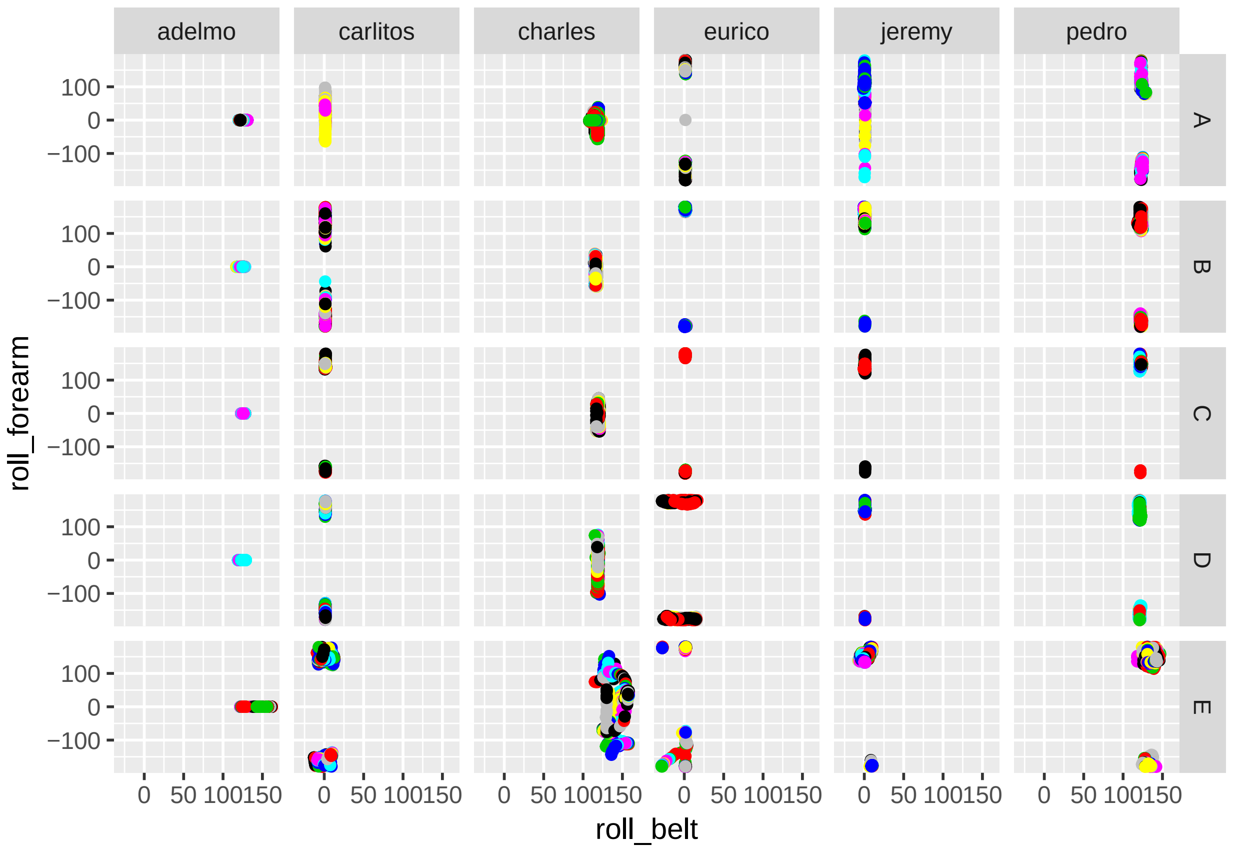
If we look at a table of our outcome variable, we notice that class A is over-represented, which may have to be dealt with by penalizing the fit on this class.

```
## [1] "Number of Observations Per Workout Form"
```

```
##
##    A    B    C    D    E
## 4464 3037 2737 2572 2885
```

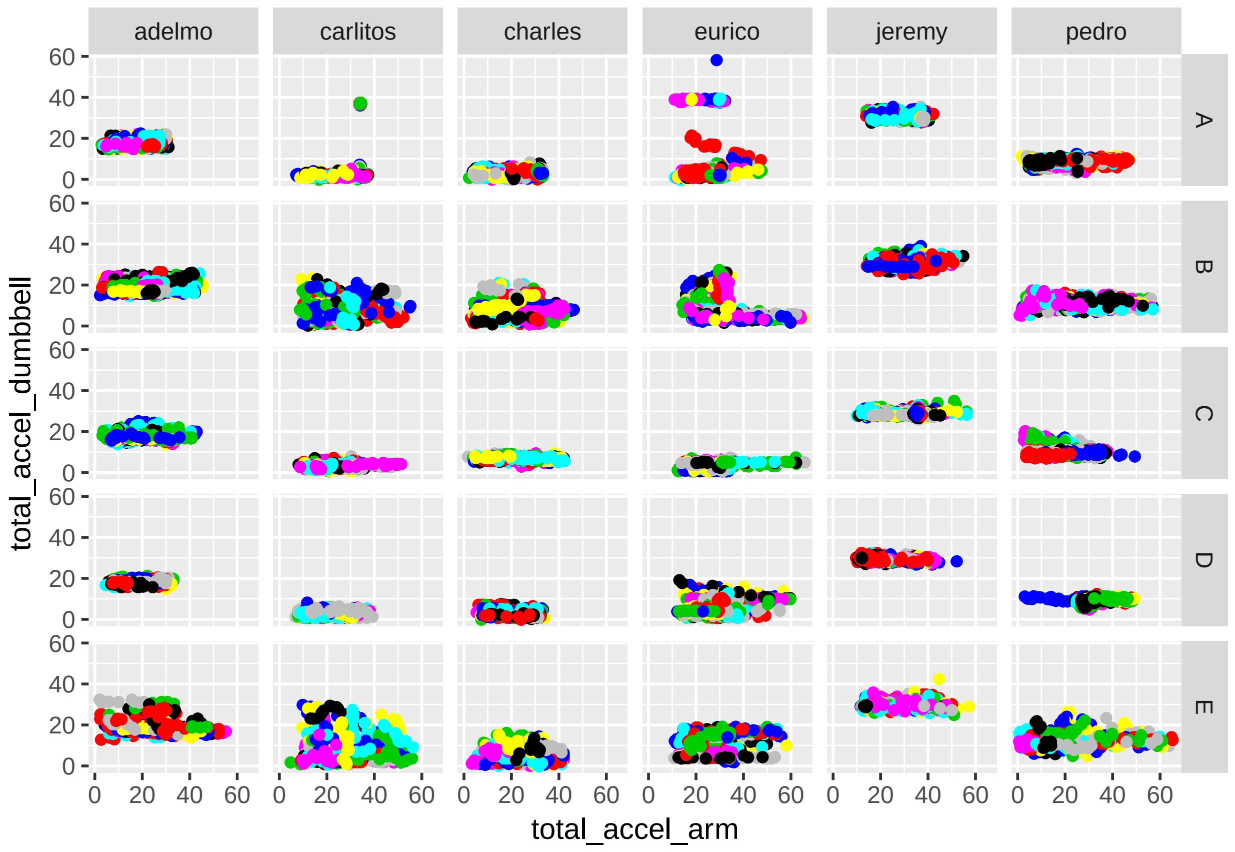
Here are some Very Hungry Caterpillar-esque graphs relating user and exersice form to a few of the various movements.

```r
ggplot(data=pretrain,aes(x=roll_belt,y=roll_forearm))+
    geom_jitter(colour=pretrain$pitch_belt+100)+facet_grid(classe~user_name)
```



Notice how little the belt_roll contributes to the class outcomes, it pretty much only varies between users and not at all between class per user.

```
ggplot(data=pretrain,aes(x=total_accel_arm,y=total_accel_dumbbell))+
    geom_jitter(colour=pretrain$raw_timestamp_part_1)+facet_grid(classe~user_name)
```



Model Selection and Feature Selection

One initial caveat is that we should not use user_name for predicting the new data. If we include user_name, the value of our model goes way down because it's not generalizable to people besides those 6 in the data set. If we want to eventually turn this model into a data product, we can't have users specify their names and hope that something about their name will give us predictive power. But in our data set, the name gives us extraordinary predictive power. Take a gander at the graphs above for the roll_forearm and belt_pitch - every user has a specific frequency associated with how they move their weight around. Of course, the variance of these is tight enough to provide enough information without user names.

Baseline Model

Our baseline model uses the chosen features and the defaults for the random forests algorithm.

```
#modelFitInitail <- train(classe~roll_belt+pitch_belt+yaw_belt+total_accel_belt+
#                              roll_arm+pitch_arm+yaw_arm+total_accel_arm+
#                              roll_dumbbell+pitch_dumbbell+yaw_dumbbell+total_accel_dumbbell+
#                              roll_forearm+pitch_forearm+yaw_forearm+total_accel_forearm, method='rf',
#                            data=train)

predsMFI <- predict(modelFitInitail,validation)
x <- confusionMatrix(predsMFI,validation$classe)

## We will not use this model below although it does very well!
## Note that this method achieved 99.88% accuracy on the validation set. While not cross-validated,
## this is pretty good for a first pass.
#modelFitInitail2 <- train(classe~user_name+roll_belt+pitch_belt+yaw_belt+total_accel_belt+
#                          roll_arm+pitch_arm+yaw_arm+total_accel_arm+
#                           roll_dumbbell+pitch_dumbbell+yaw_dumbbell+total_accel_dumbbell+
#                           roll_forearm+pitch_forearm+yaw_forearm+total_accel_forearm, method='rf',
#                         data=train)

#preds <- predict(modelFitInitail2,validation)
#y <- confusionMatrix(preds,validation$classe)
```

Our in-sample error is

```
predsMFIIS <- predict(modelFitInitail,training)
confusionMatrix(predsMFIIS,training$classe)$overall[1]
```

```
##  Accuracy
## 0.9934257
```

while our out-of-sample error for the validation set is

```
print(x$overall[1])
```

```
##  Accuracy
## 0.9844626
```

One interesting note on the model above is that the variable importance ranks the names of the users at the very end. We don't lose a ton of accuracy, and in fact, we have a pretty good model already.

Parameter search for Random Forests, Boosting, SVM, and Multinomial Regression Models

Random Forests Parameter Search

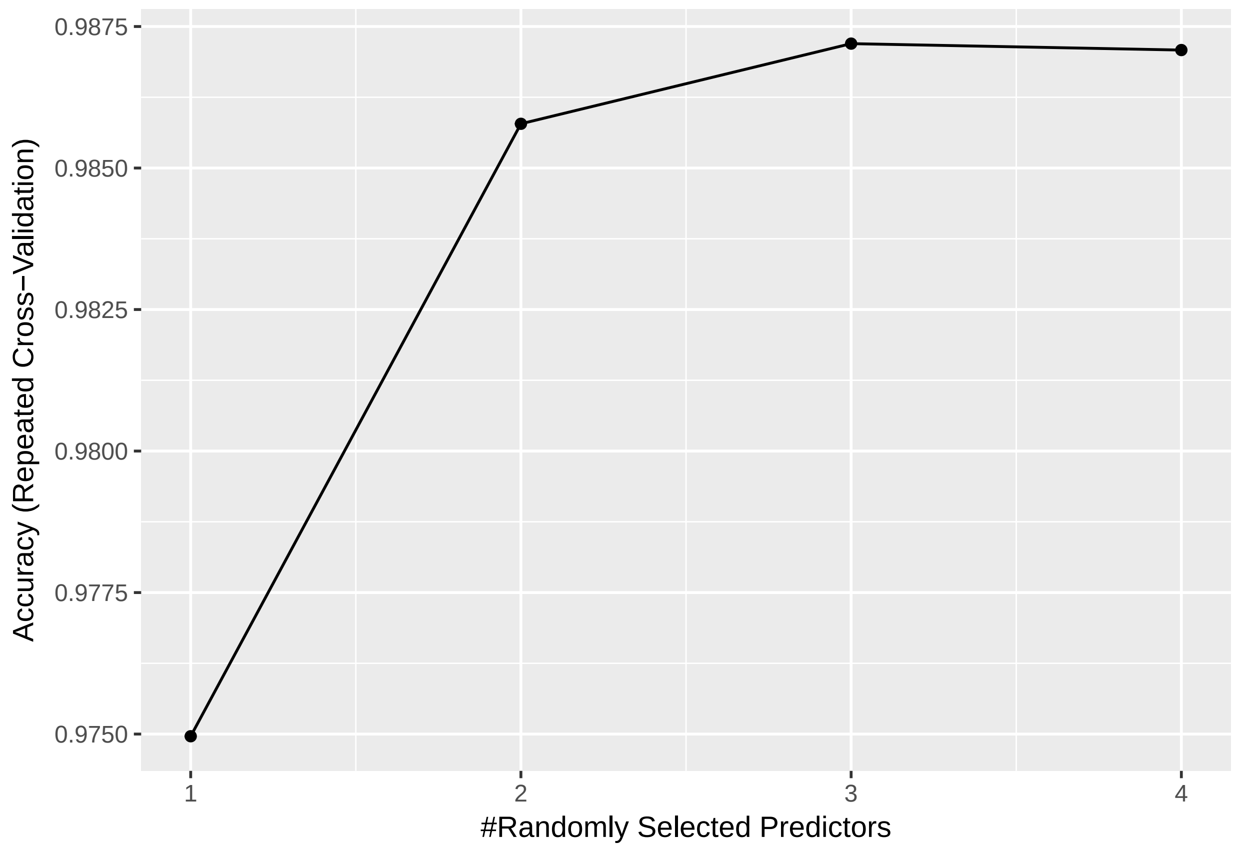
Random Forests in Caret allows us to change mtry and the number of the trees. I will leave the number of trees at default since our original model did so well. We will choose an mtry parameter which gives us the highest accuracy.

Note that we are using repeated cross-validation on the train set here, with 10 folds. We are simply trying to emulate the process by which we will eventually derive our statistic for the out-of-sample error on the test set. However, if we increased our number of repeats, we would increase our risk of overfitting. We use the validation set to make sure this didn't happen.

```
## code adapted from http://machinelearningmastery.com/tune-machine-learning-algorithms-in-r/

control <- trainControl(method="repeatedcv", number=10, repeats=3, search="grid")
grid <- expand.grid(.mtry=c(1:4))
mtry <- 4 ## This is the maximum mtry (i.e. Sqrt(16) when we have 16 features)
metric <- "Accuracy"
#rf_random2 <- train(classe~roll_belt+pitch_belt+yaw_belt+total_accel_belt+
#                           roll_arm+pitch_arm+yaw_arm+total_accel_arm+
#                           roll_dumbbell+pitch_dumbbell+yaw_dumbbell+total_accel_dumbbell+
#                          roll_forearm+pitch_forearm+yaw_forearm+total_accel_forearm, data=train, #met
print(rf_random2$results)
```

```
##   mtry  Accuracy     Kappa  AccuracySD     KappaSD
## 1    1 0.9749617 0.9683250 0.004225375 0.005350267
## 2    2 0.9857814 0.9820129 0.004005822 0.005069167
## 3    3 0.9871978 0.9838059 0.003826940 0.004841971
## 4    4 0.9870843 0.9836620 0.004148805 0.005249371
```

```
ggplot(rf_random2)
```



It appears the best parameter for Random Forests is mtry=3.

```
rfPred <- predict(rf_random, validation)
rfAccuracy <- confusionMatrix(rfPred,validation$classe)

rfAccuracy$overall
```

These are encouraging results! An out-of-sample error < in-sample error at .9883 is great.

Boosting (GBM) Parameter Search

My computer has only 8GB of RAM which makes difficult the parameter search for this particular model. I have limited the interaction depth and number of trees, but the end result may not be tenable.

```
gbmGrid <- expand.grid(.interaction.depth = 6, .n.trees = 400*(1:10),
                       .shrinkage = c(.01,.05), .n.minobsinnode=10)
```

```
gbmGrid <- expand.grid(.interaction.depth = 6, .n.trees = c(2:4*1000),
                       .shrinkage = .05, .n.minobsinnode=10)
control2 <- trainControl(method="repeatedcv", number=2, repeats=1, search="grid")
```

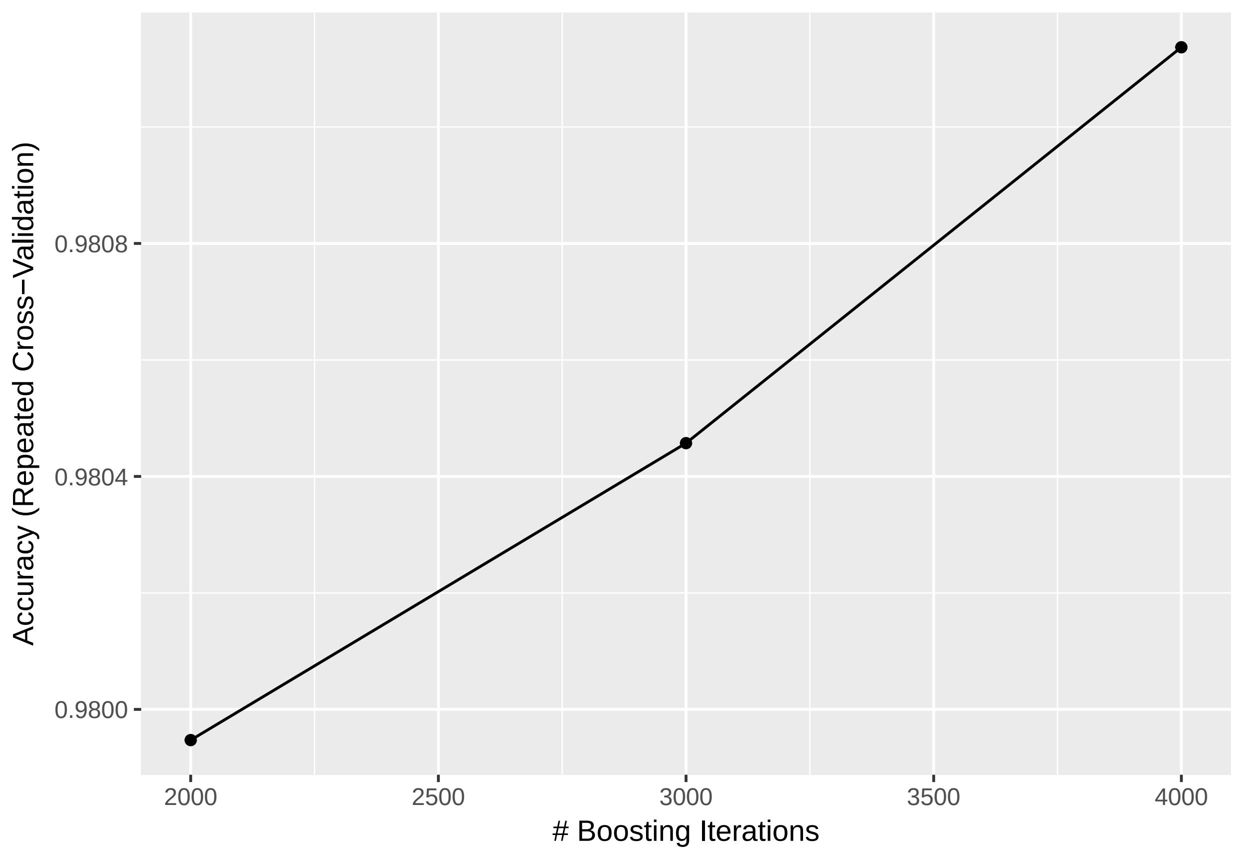
```r
#gbm_parametersearch <- train(classe~roll_belt+pitch_belt+yaw_belt+total_accel_belt+
#                             roll_arm+pitch_arm+yaw_arm+total_accel_arm+
#                             roll_dumbbell+pitch_dumbbell+yaw_dumbbell+total_accel_dumbbell+
#                             roll_forearm+pitch_forearm+yaw_forearm+total_accel_forearm, data=train, met

predgbm<- predict(gbm_parametersearch, validation)
confusionMatrix(predgbm,validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1106    2    0    1    0
##          B    7  746    5    1    1
##          C    1   10  676    0    3
##          D    1    0    4  638    6
##          E    1    2    0    3  712
##
## Overall Statistics
##
##                Accuracy : 0.9878
##                  95% CI : (0.9838, 0.991)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9845
##  Mcnemar's Test P-Value : 0.1062
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9910   0.9816   0.9869   0.9922   0.9861
## Specificity            0.9989   0.9956   0.9957   0.9966   0.9981
## Pos Pred Value         0.9973   0.9816   0.9797   0.9831   0.9916
## Neg Pred Value         0.9965   0.9956   0.9972   0.9985   0.9969
## Prevalence             0.2843   0.1936   0.1745   0.1638   0.1839
## Detection Rate         0.2817   0.1900   0.1722   0.1625   0.1814
## Detection Prevalence   0.2825   0.1936   0.1758   0.1653   0.1829
## Balanced Accuracy      0.9950   0.9886   0.9913   0.9944   0.9921
```

```r
print(gbm_parametersearch$results)
```

```
##   shrinkage interaction.depth n.minobsinnode n.trees  Accuracy     Kappa
## 1      0.05                 6             10    2000 0.9799473 0.9746383
## 2      0.05                 6             10    3000 0.9804571 0.9752835
## 3      0.05                 6             10    4000 0.9811368 0.9761434
##     AccuracySD      KappaSD
## 1 0.0002427382 0.0003071127
## 2 0.0002426769 0.0003091313
## 3 0.0007232524 0.0009153498
```

```r
ggplot(gbm_parametersearch)
```



We therefore use 4000 trees, an interaction depth of 6, and shrinkage of .05. The accuracy is good, but if we were to run the full parameter search, the model has a diminished rate of return per trees.

SVM

I haven't had good luck with svm yet, but I thought I would include it since it was covered in class. The tuning for this function is quite different from caret's built in tuning.

```r
predictors <- subset(train,select = c(roll_belt,pitch_belt,yaw_belt,total_accel_belt,
                           roll_arm,pitch_arm,yaw_arm,total_accel_arm,
                           roll_dumbbell,pitch_dumbbell,yaw_dumbbell,total_accel_dumbbell,
                           roll_forearm,pitch_forearm,yaw_forearm,total_accel_forearm))
outcome <- subset(train,select= c(classe))


svm_tune <- tune(svm, train.x=predictors, train.y=outcome$classe,
              kernel="radial", ranges=list(cost=c(1,50,100), gamma=(c(.5,1,1.5))))
```

```r
predictors.validation <- subset(validation,select = c(roll_belt,pitch_belt,yaw_belt,total_accel_belt,
                           roll_arm,pitch_arm,yaw_arm,total_accel_arm,
                           roll_dumbbell,pitch_dumbbell,yaw_dumbbell,total_accel_dumbbell,
                           roll_forearm,pitch_forearm,yaw_forearm,total_accel_forearm))

#svm_final <- svm(predictors,outcome$classe,cost=100,gamma=.5,kernal="radial")
svmpreds <- predict(svm_final, predictors.validation)
svmaccuracy <- predict(svm_final, predictors)
```

```r
confusionMatrix(svmpreds,validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1107    9    2    4    1
##          B    5  736   13    4    8
##          C    1   11  659   17    3
##          D    0    1    9  612    5
##          E    3    3    2    6  705
##
## Overall Statistics
##
##                Accuracy : 0.9727
##                  95% CI : (0.9672, 0.9776)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9655
##  Mcnemar's Test P-Value : 0.1987
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9919   0.9684   0.9620   0.9518   0.9765
## Specificity            0.9943   0.9905   0.9901   0.9954   0.9956
## Pos Pred Value         0.9858   0.9608   0.9537   0.9761   0.9805
## Neg Pred Value         0.9968   0.9924   0.9920   0.9906   0.9947
## Prevalence             0.2843   0.1936   0.1745   0.1638   0.1839
## Detection Rate         0.2820   0.1875   0.1679   0.1559   0.1796
## Detection Prevalence   0.2860   0.1951   0.1760   0.1597   0.1831
## Balanced Accuracy      0.9931   0.9795   0.9761   0.9736   0.9860
```

```r
confusionMatrix(svmaccuracy,train$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 3348    0    0    0    0
##          B    0 2274    1    0    0
##          C    0    3 2048   13    0
##          D    0    0    3 1914    2
##          E    0    0    0    2 2161
##
## Overall Statistics
##
##                Accuracy : 0.998
##                  95% CI : (0.997, 0.9987)
##     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   0.9987   0.9981   0.9922   0.9991
## Specificity            1.0000   0.9999   0.9984   0.9995   0.9998
## Pos Pred Value         1.0000   0.9996   0.9922   0.9974   0.9991
## Neg Pred Value         1.0000   0.9997   0.9996   0.9985   0.9998
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1932   0.1740   0.1626   0.1836
## Detection Prevalence   0.2845   0.1933   0.1754   0.1631   0.1838
## Balanced Accuracy      1.0000   0.9993   0.9982   0.9959   0.9994
```

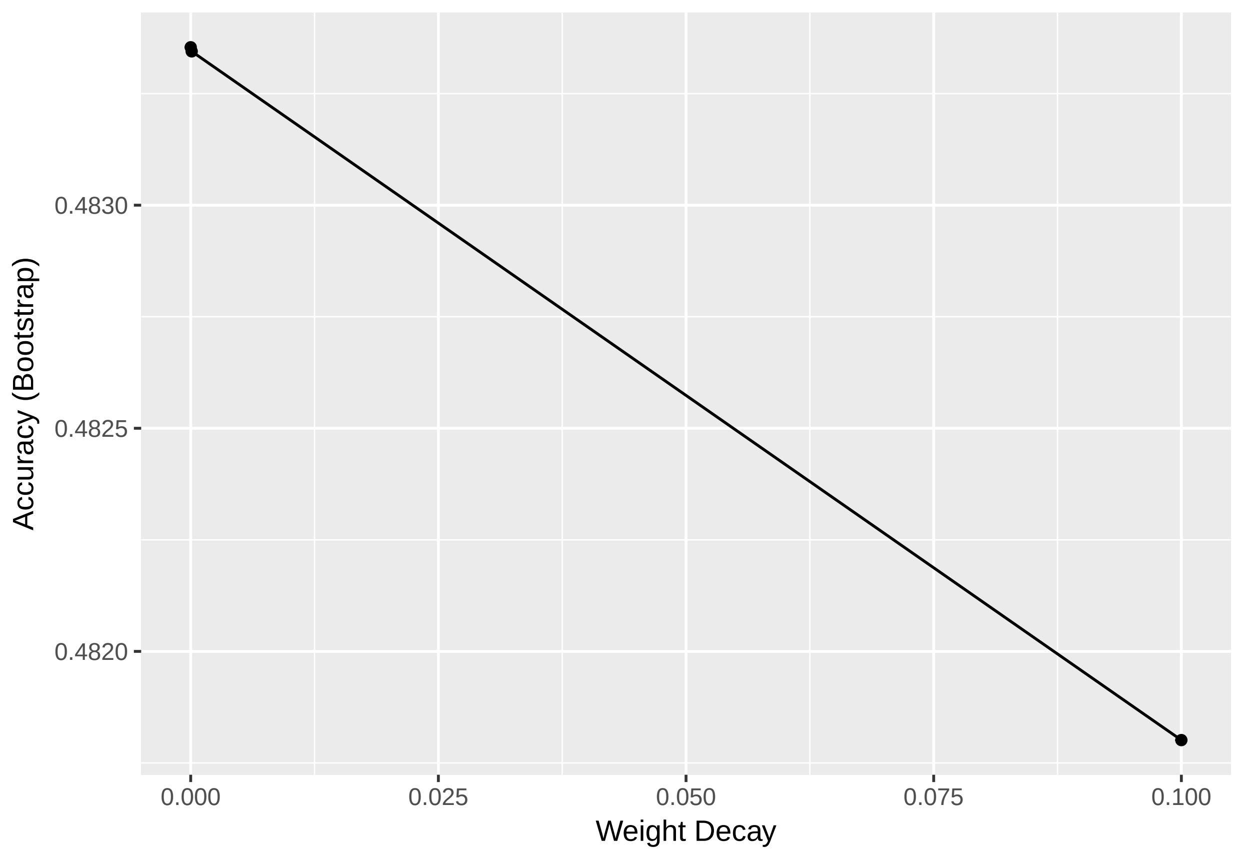
Multinomial Regression

This is an example not covered in class, but would be a traditional way to model data from 5 classification categories. From one point of view, if this model fails, how robust are our simple inferences?

Say I had two conditions which generated varying levels of the 5 classifications. I fit two multinomial models to each and compare them using some sort of inferential test - chi-square or some comparison of variance. The issue that arises is that my models may be non-robust. If they have a hard time predicting future values, I should be concerned about their inter-comparability.

```
control3 <- trainControl(method="repeatedcv", number=10, repeats=5, search="random")

mr_parametersearch <- train(classe~roll_belt+pitch_belt+yaw_belt+total_accel_belt+
                           roll_arm+pitch_arm+yaw_arm+total_accel_arm+
                           roll_dumbbell+pitch_dumbbell+yaw_dumbbell+total_accel_dumbbell+
                           roll_forearm+pitch_forearm+yaw_forearm+total_accel_forearm, data=train, meth
ggplot(mr_parametersearch)
```

```
#mr_model <- train(classe~roll_belt+pitch_belt+yaw_belt+total_accel_belt+
#                            roll_arm+pitch_arm+yaw_arm+total_accel_arm+
#                            roll_dumbbell+pitch_dumbbell+yaw_dumbbell+total_accel_dumbbell+
#                            roll_forearm+pitch_forearm+yaw_forearm+total_accel_forearm, data=train, #me

ggplot(mr_model)
```



```
predmr <- predict(mr_model,validation)
```

```
## Loading required package: nnet
```

```
confusionMatrix(predmr,validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 827 181 196  92 116
##          B  56 268  44  78 187
##          C  61  93 305  95 135
##          D 111  87  55 294 112
##          E  61 131  85  84 172
##
## Overall Statistics
##
##                Accuracy : 0.4753
##                  95% CI : (0.4596, 0.4911)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3303
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
```

```
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7410  0.35263  0.44526  0.45723  0.23823
## Specificity            0.7918  0.88471  0.88152  0.88882  0.88733
## Pos Pred Value         0.5857  0.42338  0.44267  0.44613  0.32270
## Neg Pred Value         0.8850  0.85059  0.88261  0.89317  0.83790
## Prevalence             0.2843  0.19358  0.17448  0.16378  0.18390
## Detection Rate         0.2106  0.06826  0.07769  0.07489  0.04381
## Detection Prevalence   0.3597  0.16123  0.17550  0.16786  0.13576
## Balanced Accuracy      0.7664  0.61867  0.66339  0.67303  0.56278
```

Let's just say that I'm not impressed. The weights actually brought down the accuracy. I suppose I could try to weight category A more, since it's more prominent but overall this is only marginally better than guessing.

Although Random Forests, SVM, and Boosting were all very accurate, I turn to Random Forests for the final model, since they take the least time to fit.

Estimating the Out-of-Sample Accuracy

We now have a model that seems to be well-validated and we want to establish a conjecture about its accuracy on the testing set.

I proceed to do a random 5 fold cross-validation of my final model on the "pretrain" object, our undifferentiated training set that was used to train and evaluate the model.

```
kfold<- createFolds(pretrain$classe,5)

#intialize variables
results <- data.frame(matrix(ncol=7,nrow=5))
trainfinal <- data.frame()
validationfinal <- data.frame()
grid2 <- expand.grid(.mtry=c(4))
i <- 1
#for(i in 1:5) {
#  trainfinal <- pretrain[-kfold[[i]],]
#  validationfinal <- pretrain[kfold[[i]],]

  ## build model
#  rf_random <- train(classe~roll_belt+pitch_belt+yaw_belt+total_accel_belt+
#                             roll_arm+pitch_arm+yaw_arm+total_accel_arm+
#                             roll_dumbbell+pitch_dumbbell+yaw_dumbbell+total_accel_dumbbell+
#                             roll_forearm+pitch_forearm+yaw_forearm+total_accel_forearm, data=trainfinal
#  predfinal <- predict(rf_random,validationfinal)
#  results[i,] <- confusionMatrix(predfinal,validationfinal$classe)$overall
#}

#rf_random <- train(classe~roll_belt+pitch_belt+yaw_belt+total_accel_belt+
#                             roll_arm+pitch_arm+yaw_arm+total_accel_arm+
#                             roll_dumbbell+pitch_dumbbell+yaw_dumbbell+total_accel_dumbbell+
#                             roll_forearm+pitch_forearm+yaw_forearm+total_accel_forearm, data=pretrain,

predtest <- predict(rf_random,test)
confusionMatrix(predtest,test$classe)
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1113    3    0    0    0
##          B    3  741    2    1    2
##          C    0   12  679    6    2
##          D    0    4    4  637    0
##          E    0    0    0    0  718
##
## Overall Statistics
##
##                Accuracy : 0.9901
##                  95% CI : (0.9864, 0.9929)
##     No Information Rate : 0.2842
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9874
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9973   0.9750   0.9912   0.9891   0.9945
## Specificity            0.9989   0.9975   0.9938   0.9976   1.0000
## Pos Pred Value         0.9973   0.9893   0.9714   0.9876   1.0000
## Neg Pred Value         0.9989   0.9940   0.9981   0.9979   0.9988
## Prevalence             0.2842   0.1935   0.1744   0.1640   0.1839
## Detection Rate         0.2834   0.1887   0.1729   0.1622   0.1828
## Detection Prevalence   0.2842   0.1907   0.1780   0.1642   0.1828
## Balanced Accuracy      0.9981   0.9862   0.9925   0.9933   0.9972
```

```
predholdout <- predict(rf_random,testing)

print(results[,1])
```

```
## [1] NA NA NA NA NA
```

The first column of the results represents the estimates for our model's out-of-sample accuracy. When we compare these to the final accuracy, given by the confusion matrix above it, we see that the out-of-sample accuracy was actually slightly higher, but still within .3% of the accuracy with the pretrain model.

Conclusion

With little effort, we were able to fit 3 robust models (random forests, boosting, and svm) to the WLE dataset. These models could be built into fitbit gear in the future, and give real-time advice to users about form issues. Further extensions would include more users, more forms, and different types of exersizes. Integration with apps such as MyFitnessPal could enable users to upload their own data with 'good' and 'bad' examples of exercise form, spawning a vast data lake for which the exercise community would benefit.

The multinomial regression was unfortunately innacurate and may serve as an example to dissuade others from over-interpreting such models in future studies. In addition, perhaps the model was not appropriate for the study at hand, and it would of interest to find out why this might be.