Practical Machine Learning Course Project

## 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, your 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> (see the section on the Weight Lifting Exercise Dataset).

#### Data

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 goal of this project is to predict the manner of performing unilateral dumbbell biceps curls based on data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. The 5 possible methods include - - 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

## Load the libraries

library(AppliedPredictiveModeling)  
library(caret)

## Loading required package: lattice  
## Loading required package: ggplot2

library(rattle)

## Rattle: A free graphical interface for data mining with R.  
## Version 3.4.1 Copyright (c) 2006-2014 Togaware Pty Ltd.  
## Type 'rattle()' to shake, rattle, and roll your data.

library(rpart.plot)

## Loading required package: rpart

library(randomForest)

## randomForest 4.6-10  
## Type rfNews() to see new features/changes/bug fixes.

### Load input data

# Download data.  
url\_raw\_training <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"  
file\_dest\_training <- "pml-training.csv"  
  
url\_raw\_testing <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"  
file\_dest\_testing <- "pml-testing.csv"  
  
  
# Import the data treating empty values as NA.  
df\_training <- read.csv(file\_dest\_training, na.strings=c("NA",""), header=TRUE)  
colnames\_train <- colnames(df\_training)  
df\_testing <- read.csv(file\_dest\_testing, na.strings=c("NA",""), header=TRUE)  
colnames\_test <- colnames(df\_testing)  
  
# Verify that the column names (excluding classe and problem\_id) are identical in the training and test set.  
all.equal(colnames\_train[1:length(colnames\_train)-1], colnames\_test[1:length(colnames\_train)-1])

## [1] TRUE

# Count the number of non-NAs in each col.  
nonNAs <- function(x) {  
 as.vector(apply(x, 2, function(x) length(which(!is.na(x)))))  
}  
  
# Build vector of missing data or NA columns to drop.  
colcnts <- nonNAs(df\_training)  
drops <- c()  
for (cnt in 1:length(colcnts)) {  
 if (colcnts[cnt] < nrow(df\_training)) {  
 drops <- c(drops, colnames\_train[cnt])  
 }  
}  
  
# Drop NA data and the first 7 columns as they're unnecessary for predicting.  
df\_training <- df\_training[,!(names(df\_training) %in% drops)]  
df\_training <- df\_training[,8:length(colnames(df\_training))]  
  
df\_testing <- df\_testing[,!(names(df\_testing) %in% drops)]  
df\_testing <- df\_testing[,8:length(colnames(df\_testing))]  
  
# Show remaining columns.  
colnames(df\_training)

## [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"

colnames(df\_testing)

## [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" "problem\_id"

check for covariates that have virtually no variablility.

nsv <- nearZeroVar(df\_training, saveMetrics=TRUE)  
nsv

## freqRatio percentUnique zeroVar nzv  
## roll\_belt 1.101904 6.7781062 FALSE FALSE  
## pitch\_belt 1.036082 9.3772296 FALSE FALSE  
## yaw\_belt 1.058480 9.9734991 FALSE FALSE  
## total\_accel\_belt 1.063160 0.1477933 FALSE FALSE  
## gyros\_belt\_x 1.058651 0.7134849 FALSE FALSE  
## gyros\_belt\_y 1.144000 0.3516461 FALSE FALSE  
## gyros\_belt\_z 1.066214 0.8612782 FALSE FALSE  
## accel\_belt\_x 1.055412 0.8357966 FALSE FALSE  
## accel\_belt\_y 1.113725 0.7287738 FALSE FALSE  
## accel\_belt\_z 1.078767 1.5237998 FALSE FALSE  
## magnet\_belt\_x 1.090141 1.6664968 FALSE FALSE  
## magnet\_belt\_y 1.099688 1.5187035 FALSE FALSE  
## magnet\_belt\_z 1.006369 2.3290184 FALSE FALSE  
## roll\_arm 52.338462 13.5256345 FALSE FALSE  
## pitch\_arm 87.256410 15.7323412 FALSE FALSE  
## yaw\_arm 33.029126 14.6570176 FALSE FALSE  
## total\_accel\_arm 1.024526 0.3363572 FALSE FALSE  
## gyros\_arm\_x 1.015504 3.2769341 FALSE FALSE  
## gyros\_arm\_y 1.454369 1.9162165 FALSE FALSE  
## gyros\_arm\_z 1.110687 1.2638875 FALSE FALSE  
## accel\_arm\_x 1.017341 3.9598410 FALSE FALSE  
## accel\_arm\_y 1.140187 2.7367241 FALSE FALSE  
## accel\_arm\_z 1.128000 4.0362858 FALSE FALSE  
## magnet\_arm\_x 1.000000 6.8239731 FALSE FALSE  
## magnet\_arm\_y 1.056818 4.4439914 FALSE FALSE  
## magnet\_arm\_z 1.036364 6.4468454 FALSE FALSE  
## roll\_dumbbell 1.022388 84.2065029 FALSE FALSE  
## pitch\_dumbbell 2.277372 81.7449801 FALSE FALSE  
## yaw\_dumbbell 1.132231 83.4828254 FALSE FALSE  
## total\_accel\_dumbbell 1.072634 0.2191418 FALSE FALSE  
## gyros\_dumbbell\_x 1.003268 1.2282132 FALSE FALSE  
## gyros\_dumbbell\_y 1.264957 1.4167771 FALSE FALSE  
## gyros\_dumbbell\_z 1.060100 1.0498420 FALSE FALSE  
## accel\_dumbbell\_x 1.018018 2.1659362 FALSE FALSE  
## accel\_dumbbell\_y 1.053061 2.3748853 FALSE FALSE  
## accel\_dumbbell\_z 1.133333 2.0894914 FALSE FALSE  
## magnet\_dumbbell\_x 1.098266 5.7486495 FALSE FALSE  
## magnet\_dumbbell\_y 1.197740 4.3012945 FALSE FALSE  
## magnet\_dumbbell\_z 1.020833 3.4451126 FALSE FALSE  
## roll\_forearm 11.589286 11.0895933 FALSE FALSE  
## pitch\_forearm 65.983051 14.8557741 FALSE FALSE  
## yaw\_forearm 15.322835 10.1467740 FALSE FALSE  
## total\_accel\_forearm 1.128928 0.3567424 FALSE FALSE  
## gyros\_forearm\_x 1.059273 1.5187035 FALSE FALSE  
## gyros\_forearm\_y 1.036554 3.7763735 FALSE FALSE  
## gyros\_forearm\_z 1.122917 1.5645704 FALSE FALSE  
## accel\_forearm\_x 1.126437 4.0464784 FALSE FALSE  
## accel\_forearm\_y 1.059406 5.1116094 FALSE FALSE  
## accel\_forearm\_z 1.006250 2.9558659 FALSE FALSE  
## magnet\_forearm\_x 1.012346 7.7667924 FALSE FALSE  
## magnet\_forearm\_y 1.246914 9.5403119 FALSE FALSE  
## magnet\_forearm\_z 1.000000 8.5771073 FALSE FALSE  
## classe 1.469581 0.0254816 FALSE FALSE

Given that all of the near zero variance variables (nsv) are FALSE, there's no need to eliminate any covariates due to lack of variablility.

#### Partition the data

# Divide the given training set into 4 roughly equal sets.  
set.seed(666)  
ids\_small <- createDataPartition(y=df\_training$classe, p=0.25, list=FALSE)  
df\_small1 <- df\_training[ids\_small,]  
df\_remainder <- df\_training[-ids\_small,]  
set.seed(666)  
ids\_small <- createDataPartition(y=df\_remainder$classe, p=0.33, list=FALSE)  
df\_small2 <- df\_remainder[ids\_small,]  
df\_remainder <- df\_remainder[-ids\_small,]  
set.seed(666)  
ids\_small <- createDataPartition(y=df\_remainder$classe, p=0.5, list=FALSE)  
df\_small3 <- df\_remainder[ids\_small,]  
df\_small4 <- df\_remainder[-ids\_small,]  
# Divide each of these 4 sets into training (60%) and test (40%) sets.  
set.seed(666)  
inTrain <- createDataPartition(y=df\_small1$classe, p=0.6, list=FALSE)  
df\_small\_training1 <- df\_small1[inTrain,]  
df\_small\_testing1 <- df\_small1[-inTrain,]  
set.seed(666)  
inTrain <- createDataPartition(y=df\_small2$classe, p=0.6, list=FALSE)  
df\_small\_training2 <- df\_small2[inTrain,]  
df\_small\_testing2 <- df\_small2[-inTrain,]  
set.seed(666)  
inTrain <- createDataPartition(y=df\_small3$classe, p=0.6, list=FALSE)  
df\_small\_training3 <- df\_small3[inTrain,]  
df\_small\_testing3 <- df\_small3[-inTrain,]  
set.seed(666)  
inTrain <- createDataPartition(y=df\_small4$classe, p=0.6, list=FALSE)  
df\_small\_training4 <- df\_small4[inTrain,]  
df\_small\_testing4 <- df\_small4[-inTrain,]

Choosen two different algorithms.

### Prediction Evaluation

#### Classification Tree

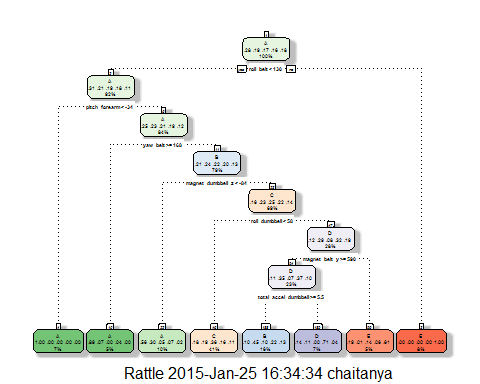
# Train on training set 1 of 4 with no extra features.  
set.seed(666)  
modFit <- train(df\_small\_training1$classe ~ ., data = df\_small\_training1, method="rpart")  
print(modFit, digits=3)

## CART   
##   
## 2946 samples  
## 52 predictor  
## 5 classes: 'A', 'B', 'C', 'D', 'E'   
##   
## No pre-processing  
## Resampling: Bootstrapped (25 reps)   
##   
## Summary of sample sizes: 2946, 2946, 2946, 2946, 2946, 2946, ...   
##   
## Resampling results across tuning parameters:  
##   
## cp Accuracy Kappa Accuracy SD Kappa SD  
## 0.0346 0.531 0.4003 0.0355 0.0479   
## 0.0442 0.471 0.3076 0.0555 0.0967   
## 0.1162 0.324 0.0602 0.0456 0.0641   
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was cp = 0.0346.

print(modFit$finalModel, digits=3)

## n= 2946   
##   
## node), split, n, loss, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 2946 2110 A (0.28 0.19 0.17 0.16 0.18)   
## 2) roll\_belt< 130 2699 1860 A (0.31 0.21 0.19 0.18 0.11)   
## 4) pitch\_forearm< -34 220 0 A (1 0 0 0 0) \*  
## 5) pitch\_forearm>=-34 2479 1860 A (0.25 0.23 0.21 0.19 0.12)   
## 10) yaw\_belt>=168 138 15 A (0.89 0.072 0 0.036 0) \*  
## 11) yaw\_belt< 168 2341 1780 B (0.21 0.24 0.22 0.2 0.13)   
## 22) magnet\_dumbbell\_z< -83.5 305 134 A (0.56 0.3 0.046 0.069 0.02) \*  
## 23) magnet\_dumbbell\_z>=-83.5 2036 1540 C (0.16 0.23 0.25 0.22 0.14)   
## 46) roll\_dumbbell< 57.7 1209 776 C (0.18 0.19 0.36 0.16 0.11) \*  
## 47) roll\_dumbbell>=57.7 827 565 D (0.12 0.29 0.081 0.32 0.19)   
## 94) magnet\_belt\_y>=590 687 433 D (0.11 0.35 0.07 0.37 0.1)   
## 188) total\_accel\_dumbbell>=5.5 474 260 B (0.097 0.45 0.1 0.22 0.13) \*  
## 189) total\_accel\_dumbbell< 5.5 213 62 D (0.14 0.11 0 0.71 0.042) \*  
## 95) magnet\_belt\_y< 590 140 55 E (0.19 0.014 0.14 0.057 0.61) \*  
## 3) roll\_belt>=130 247 1 E (0.004 0 0 0 1) \*

fancyRpartPlot(modFit$finalModel)



# Run against testing set 1 of 4 with no extra features.  
predictions <- predict(modFit, newdata=df\_small\_testing1)  
print(confusionMatrix(predictions, df\_small\_testing1$classe), digits=4)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 368 74 11 28 8  
## B 24 151 25 83 30  
## C 135 148 288 138 99  
## D 15 7 0 69 4  
## E 16 0 18 3 219  
##   
## Overall Statistics  
##   
## Accuracy : 0.5584   
## 95% CI : (0.5361, 0.5805)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.4441   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.6595 0.3974 0.8421 0.21495 0.6083  
## Specificity 0.9138 0.8975 0.6788 0.98415 0.9769  
## Pos Pred Value 0.7526 0.4824 0.3564 0.72632 0.8555  
## Neg Pred Value 0.8709 0.8610 0.9532 0.86495 0.9173  
## Prevalence 0.2845 0.1938 0.1744 0.16369 0.1836  
## Detection Rate 0.1877 0.0770 0.1469 0.03519 0.1117  
## Detection Prevalence 0.2494 0.1596 0.4120 0.04844 0.1305  
## Balanced Accuracy 0.7866 0.6475 0.7605 0.59955 0.7926

# Train on training set 1 of 4 with only preprocessing.  
set.seed(666)  
modFit <- train(df\_small\_training1$classe ~ ., preProcess=c("center", "scale"), data = df\_small\_training1, method="rpart")  
print(modFit, digits=3)

## CART   
##   
## 2946 samples  
## 52 predictor  
## 5 classes: 'A', 'B', 'C', 'D', 'E'   
##   
## Pre-processing: centered, scaled   
## Resampling: Bootstrapped (25 reps)   
##   
## Summary of sample sizes: 2946, 2946, 2946, 2946, 2946, 2946, ...   
##   
## Resampling results across tuning parameters:  
##   
## cp Accuracy Kappa Accuracy SD Kappa SD  
## 0.0346 0.531 0.4003 0.0355 0.0479   
## 0.0442 0.471 0.3077 0.0555 0.0968   
## 0.1162 0.324 0.0602 0.0456 0.0641   
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was cp = 0.0346.

# Train on training set 1 of 4 with only cross validation.  
set.seed(666)  
modFit <- train(df\_small\_training1$classe ~ ., trControl=trainControl(method = "cv", number = 4), data = df\_small\_training1, method="rpart")  
print(modFit, digits=3)

## CART   
##   
## 2946 samples  
## 52 predictor  
## 5 classes: 'A', 'B', 'C', 'D', 'E'   
##   
## No pre-processing  
## Resampling: Cross-Validated (4 fold)   
##   
## Summary of sample sizes: 2212, 2209, 2208, 2209   
##   
## Resampling results across tuning parameters:  
##   
## cp Accuracy Kappa Accuracy SD Kappa SD  
## 0.0346 0.552 0.4266 0.0383 0.0542   
## 0.0442 0.470 0.3041 0.0689 0.1197   
## 0.1162 0.344 0.0914 0.0405 0.0610   
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was cp = 0.0346.

#Train on training set 1 of 4 with both preprocessing and cross validation.  
set.seed(666)  
modFit <- train(df\_small\_training1$classe ~ ., preProcess=c("center", "scale"), trControl=trainControl(method = "cv", number = 4), data = df\_small\_training1, method="rpart")  
print(modFit, digits=3)

## CART   
##   
## 2946 samples  
## 52 predictor  
## 5 classes: 'A', 'B', 'C', 'D', 'E'   
##   
## Pre-processing: centered, scaled   
## Resampling: Cross-Validated (4 fold)   
##   
## Summary of sample sizes: 2212, 2209, 2208, 2209   
##   
## Resampling results across tuning parameters:  
##   
## cp Accuracy Kappa Accuracy SD Kappa SD  
## 0.0346 0.552 0.4266 0.0383 0.0542   
## 0.0442 0.470 0.3041 0.0689 0.1197   
## 0.1162 0.344 0.0914 0.0405 0.0610   
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was cp = 0.0346.

# Run against testing set 1 of 4 with both preprocessing and cross validation.  
predictions <- predict(modFit, newdata=df\_small\_testing1)  
print(confusionMatrix(predictions, df\_small\_testing1$classe), digits=4)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 368 74 11 28 8  
## B 24 151 25 83 30  
## C 135 148 288 138 99  
## D 15 7 0 69 4  
## E 16 0 18 3 219  
##   
## Overall Statistics  
##   
## Accuracy : 0.5584   
## 95% CI : (0.5361, 0.5805)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.4441   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.6595 0.3974 0.8421 0.21495 0.6083  
## Specificity 0.9138 0.8975 0.6788 0.98415 0.9769  
## Pos Pred Value 0.7526 0.4824 0.3564 0.72632 0.8555  
## Neg Pred Value 0.8709 0.8610 0.9532 0.86495 0.9173  
## Prevalence 0.2845 0.1938 0.1744 0.16369 0.1836  
## Detection Rate 0.1877 0.0770 0.1469 0.03519 0.1117  
## Detection Prevalence 0.2494 0.1596 0.4120 0.04844 0.1305  
## Balanced Accuracy 0.7866 0.6475 0.7605 0.59955 0.7926

The impact of incorporating both preprocessing and cross validation appeared to show some minimal improvement (accuracy rate rose from 0.531 to 0.552 against training sets). However, when run against the corresponding testing set, the accuracy rate was identical (0.5584) for both the "out of the box" and the preprocessing/cross validation methods.

#### Random Forest

# Train on training set 1 of 4 with only cross validation.  
set.seed(666)  
modFit <- train(df\_small\_training1$classe ~ ., method="rf", trControl=trainControl(method = "cv", number = 4), data=df\_small\_training1)  
print(modFit, digits=3)

## Random Forest   
##   
## 2946 samples  
## 52 predictor  
## 5 classes: 'A', 'B', 'C', 'D', 'E'   
##   
## No pre-processing  
## Resampling: Cross-Validated (4 fold)   
##   
## Summary of sample sizes: 2212, 2209, 2208, 2209   
##   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa Accuracy SD Kappa SD  
## 2 0.951 0.939 0.00449 0.00570   
## 27 0.955 0.943 0.00582 0.00736   
## 52 0.951 0.938 0.00888 0.01121   
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 27.

# Run against testing set 1 of 4.  
predictions <- predict(modFit, newdata=df\_small\_testing1)  
print(confusionMatrix(predictions, df\_small\_testing1$classe), digits=4)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 555 12 1 0 1  
## B 2 358 12 1 0  
## C 0 9 324 6 4  
## D 0 1 5 309 1  
## E 1 0 0 5 354  
##   
## Overall Statistics  
##   
## Accuracy : 0.9689   
## 95% CI : (0.9602, 0.9761)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9606   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.9946 0.9421 0.9474 0.9626 0.9833  
## Specificity 0.9900 0.9905 0.9883 0.9957 0.9963  
## Pos Pred Value 0.9754 0.9598 0.9446 0.9778 0.9833  
## Neg Pred Value 0.9978 0.9861 0.9889 0.9927 0.9963  
## Prevalence 0.2845 0.1938 0.1744 0.1637 0.1836  
## Detection Rate 0.2830 0.1826 0.1652 0.1576 0.1805  
## Detection Prevalence 0.2902 0.1902 0.1749 0.1611 0.1836  
## Balanced Accuracy 0.9923 0.9663 0.9678 0.9792 0.9898

# Run against 20 testing set provided by Professor Leek.  
print(predict(modFit, newdata=df\_testing))

## [1] B A A A A E D B A A B C B A E E A B B B  
## Levels: A B C D E

# Train on training set 1 of 4 with only both preprocessing and cross validation.  
set.seed(666)  
modFit <- train(df\_small\_training1$classe ~ ., method="rf", preProcess=c("center", "scale"), trControl=trainControl(method = "cv", number = 4), data=df\_small\_training1)  
print(modFit, digits=3)

## Random Forest   
##   
## 2946 samples  
## 52 predictor  
## 5 classes: 'A', 'B', 'C', 'D', 'E'   
##   
## Pre-processing: centered, scaled   
## Resampling: Cross-Validated (4 fold)   
##   
## Summary of sample sizes: 2212, 2209, 2208, 2209   
##   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa Accuracy SD Kappa SD  
## 2 0.951 0.939 0.00382 0.00482   
## 27 0.954 0.942 0.00466 0.00590   
## 52 0.952 0.939 0.01066 0.01347   
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 27.

# Run against testing set 1 of 4.  
predictions <- predict(modFit, newdata=df\_small\_testing1)  
print(confusionMatrix(predictions, df\_small\_testing1$classe), digits=4)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 555 10 0 0 0  
## B 2 357 11 0 0  
## C 0 12 327 6 5  
## D 0 1 4 312 1  
## E 1 0 0 3 354  
##   
## Overall Statistics  
##   
## Accuracy : 0.9714   
## 95% CI : (0.9631, 0.9784)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9639   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.9946 0.9395 0.9561 0.9720 0.9833  
## Specificity 0.9929 0.9918 0.9858 0.9963 0.9975  
## Pos Pred Value 0.9823 0.9649 0.9343 0.9811 0.9888  
## Neg Pred Value 0.9979 0.9855 0.9907 0.9945 0.9963  
## Prevalence 0.2845 0.1938 0.1744 0.1637 0.1836  
## Detection Rate 0.2830 0.1820 0.1668 0.1591 0.1805  
## Detection Prevalence 0.2881 0.1887 0.1785 0.1622 0.1826  
## Balanced Accuracy 0.9937 0.9656 0.9710 0.9842 0.9904

# Run against 20 testing set provided by Professor Leek.  
print(predict(modFit, newdata=df\_testing))

## [1] B A A A A E D B A A B C B A E E A B B B  
## Levels: A B C D E

Preprocessing actually lowered the accuracy rate from 0.955 to 0.954 against the training set. However, when run against the corresponding set, the accuracy rate rose from 0.9689 to 0.9714 with the addition of preprocessing. Thus I decided to apply both preprocessing and cross validation to the remaining 3 data sets.

# Train on training set 2 of 4 with only cross validation.  
set.seed(666)  
modFit <- train(df\_small\_training2$classe ~ ., method="rf", preProcess=c("center", "scale"), trControl=trainControl(method = "cv", number = 4), data=df\_small\_training2)  
print(modFit, digits=3)

## Random Forest   
##   
## 2917 samples  
## 52 predictor  
## 5 classes: 'A', 'B', 'C', 'D', 'E'   
##   
## Pre-processing: centered, scaled   
## Resampling: Cross-Validated (4 fold)   
##   
## Summary of sample sizes: 2188, 2188, 2187, 2188   
##   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa Accuracy SD Kappa SD  
## 2 0.952 0.939 0.00665 0.00844   
## 27 0.954 0.941 0.01023 0.01300   
## 52 0.944 0.929 0.00579 0.00735   
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 27.

# Run against testing set 2 of 4.  
predictions <- predict(modFit, newdata=df\_small\_testing2)  
print(confusionMatrix(predictions, df\_small\_testing2$classe), digits=4)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 548 11 0 2 0  
## B 3 355 14 1 5  
## C 0 9 323 10 6  
## D 0 1 1 303 5  
## E 1 0 0 2 341  
##   
## Overall Statistics  
##   
## Accuracy : 0.9634   
## 95% CI : (0.9541, 0.9713)  
## No Information Rate : 0.2844   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9537   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.9928 0.9441 0.9556 0.9528 0.9552  
## Specificity 0.9906 0.9853 0.9844 0.9957 0.9981  
## Pos Pred Value 0.9768 0.9392 0.9282 0.9774 0.9913  
## Neg Pred Value 0.9971 0.9866 0.9906 0.9908 0.9900  
## Prevalence 0.2844 0.1937 0.1741 0.1638 0.1839  
## Detection Rate 0.2823 0.1829 0.1664 0.1561 0.1757  
## Detection Prevalence 0.2890 0.1947 0.1793 0.1597 0.1772  
## Balanced Accuracy 0.9917 0.9647 0.9700 0.9743 0.9766

# Run against 20 testing set provided by Professor Leek.  
print(predict(modFit, newdata=df\_testing))

## [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

# Train on training set 3 of 4 with only cross validation.  
set.seed(666)  
modFit <- train(df\_small\_training3$classe ~ ., method="rf", preProcess=c("center", "scale"), trControl=trainControl(method = "cv", number = 4), data=df\_small\_training3)  
print(modFit, digits=3)

## Random Forest   
##   
## 2960 samples  
## 52 predictor  
## 5 classes: 'A', 'B', 'C', 'D', 'E'   
##   
## Pre-processing: centered, scaled   
## Resampling: Cross-Validated (4 fold)   
##   
## Summary of sample sizes: 2219, 2221, 2220, 2220   
##   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa Accuracy SD Kappa SD  
## 2 0.949 0.935 0.00696 0.0088   
## 27 0.951 0.938 0.01046 0.0132   
## 52 0.944 0.929 0.01156 0.0146   
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 27.

# Run against testing set 3 of 4.  
predictions <- predict(modFit, newdata=df\_small\_testing3)  
print(confusionMatrix(predictions, df\_small\_testing3$classe), digits=4)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 556 10 0 1 0  
## B 1 357 17 0 4  
## C 1 12 322 7 3  
## D 1 2 2 313 1  
## E 1 0 3 2 354  
##   
## Overall Statistics  
##   
## Accuracy : 0.9655   
## 95% CI : (0.9564, 0.9731)  
## No Information Rate : 0.2843   
## P-Value [Acc > NIR] : < 2e-16   
##   
## Kappa : 0.9563   
## Mcnemar's Test P-Value : 0.03619   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.9929 0.9370 0.9360 0.9690 0.9779  
## Specificity 0.9922 0.9862 0.9859 0.9964 0.9963  
## Pos Pred Value 0.9806 0.9420 0.9333 0.9812 0.9833  
## Neg Pred Value 0.9971 0.9849 0.9865 0.9939 0.9950  
## Prevalence 0.2843 0.1934 0.1746 0.1640 0.1838  
## Detection Rate 0.2822 0.1812 0.1635 0.1589 0.1797  
## Detection Prevalence 0.2878 0.1924 0.1751 0.1619 0.1827  
## Balanced Accuracy 0.9925 0.9616 0.9610 0.9827 0.9871

# Run against 20 testing set provided by Professor Leek.  
print(predict(modFit, newdata=df\_testing))

## [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

# Train on training set 4 of 4 with only cross validation.  
set.seed(666)  
modFit <- train(df\_small\_training4$classe ~ ., method="rf", preProcess=c("center", "scale"), trControl=trainControl(method = "cv", number = 4), data=df\_small\_training4)  
print(modFit, digits=3)

## Random Forest   
##   
## 2958 samples  
## 52 predictor  
## 5 classes: 'A', 'B', 'C', 'D', 'E'   
##   
## Pre-processing: centered, scaled   
## Resampling: Cross-Validated (4 fold)   
##   
## Summary of sample sizes: 2218, 2219, 2219, 2218   
##   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa Accuracy SD Kappa SD  
## 2 0.950 0.937 0.00656 0.00834   
## 27 0.955 0.943 0.00891 0.01128   
## 52 0.947 0.932 0.01013 0.01284   
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 27.

# Run against testing set 4 of 4.  
predictions <- predict(modFit, newdata=df\_small\_testing4)  
print(confusionMatrix(predictions, df\_small\_testing4$classe), digits=4)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 553 20 0 0 0  
## B 4 357 19 3 3  
## C 2 4 315 7 7  
## D 1 0 9 312 6  
## E 0 0 0 1 346  
##   
## Overall Statistics  
##   
## Accuracy : 0.9563   
## 95% CI : (0.9463, 0.9649)  
## No Information Rate : 0.2844   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9447   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.9875 0.9370 0.9184 0.9659 0.9558  
## Specificity 0.9858 0.9817 0.9877 0.9903 0.9994  
## Pos Pred Value 0.9651 0.9249 0.9403 0.9512 0.9971  
## Neg Pred Value 0.9950 0.9848 0.9829 0.9933 0.9901  
## Prevalence 0.2844 0.1935 0.1742 0.1640 0.1838  
## Detection Rate 0.2809 0.1813 0.1600 0.1585 0.1757  
## Detection Prevalence 0.2910 0.1960 0.1701 0.1666 0.1762  
## Balanced Accuracy 0.9867 0.9594 0.9530 0.9781 0.9776

# Run against 20 testing set provided by Professor Leek.  
print(predict(modFit, newdata=df\_testing))

## [1] B A B A A E D D A A B C B A E E A B B B  
## Levels: A B C D E

Three separate predictions by appling the 4 models against the actual 20 item training set:

1. Accuracy Rate 0.0286 Predictions: B A A A A E D B A A B C B A E E A B B B
2. Accuracy Rates 0.0366 and 0.0345 Predictions: B A B A A E D B A A B C B A E E A B B B
3. Accuracy Rate 0.0437 Predictions: B A B A A E D D A A B C B A E E A B B B

Since Professor Leek is allowing 2 submissions for each problem, I decided to attempt with the two most likely prediction sets: option A and option B.

Since options A and B above only differed for item 3 (A for option A, B for option B), I subimitted one value for problems 1-2 and 4-20, while I submitted two values for problem 3. For problem 3, I was expecting the automated grader to tell me which answer (A or B) was correct, but instead the grader simply told me I had a correct answer. All other answers were also correct, resulting in a score of 100%.