title: "Practical Machine Learning Project" author: "Ajay Rathi" date: "23 August 2015" output: html_document

---

SYNOPSIS

Source:

Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013.

Goal of this project is to "predict the manner in which trainers did the exercise."

Further report should describe:

"how you built your model" "how you used cross validation" "what you think the expected out of sample error is" "why you made the choices you did" Ultimately, the prediction model is to be run on the test data to predict the outcome of 20 different test cases.

First, though, I'll load the appropriate packages and set the seed for reproduceable results.

```
library(AppliedPredictiveModeling)

## Warning: package 'AppliedPredictiveModeling' was built under R version
## 3.2.2

library(caret)

## Warning: package 'caret' was built under R version 3.2.2

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

## Warning: package 'ggplot2' was built under R version 3.2.2

library(rattle)

## Warning: package 'rattle' was built under R version 3.2.2

## Loading required package: RGtk2

## Warning: package 'RGtk2' was built under R version 3.2.2

## Rattle: A free graphical interface for data mining with R.
## Version 3.5.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 3.2.2

## Loading required package: rpart

library(randomForest)

## Warning: package 'randomForest' was built under R version 3.2.2

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

QUESTION

In the aforementioned study, six participants participated in a dumbell lifting exercise five different ways. The five ways, as described in the study, were "exactly according to the specification (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."

By processing data gathered from accelerometers on the belt, forearm, arm, and dumbell of the participants in a machine learning algorithm, the question is can the appropriate activity quality (class A-E) be predicted?

INPUT DATA

The first step is to import the data and to verify that the training data and the test data are identical.

```
# Download data.
#url_raw_training <-
"https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
file_dest_training <- "pml-training.csv"
#download.file(url=url_raw_training, destfile=file_dest_training,
method="curl")
#url_raw_testing <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-
testing.csv"
file_dest_testing <- "pml-testing.csv"
#download.file(url=url_raw_testing, destfile=file_dest_testing,
method="curl")

# 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
```

FEATURES

Having verified that the schema of both the training and testing sets are identical (excluding the final column representing the A-E class), I decided to eliminate both NA columns and other extraneous columns.

```
# 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"
```

First, 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.

ALGORITHM

We were provided with a large training set (19,622 entries) and a small testing set (20 entries). Instead of performing the algorithm on the entire training set, as it would be time consuming and wouldn't allow for an attempt on a testing set, I chose to divide the given training set into four roughly equal sets, each of which was then split into a training set (comprising 60% of the entries) and a testing set (comprising 40% of the entries).

```
# 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,]
```

PARAMETERS

I decided to try classification trees "out of the box" and then introduce preprocessing and cross validation.

While I also considered applying "out of the box" random forest models, some of the horror stories contributed to the coursera discussion forums regarding the lengthy processing times for random forest models convinced me to only attempt random forests with cross validation and, possibly, preprocessing.

EVALUATION

Classification Tree

First, the "out of the box" classification tree:

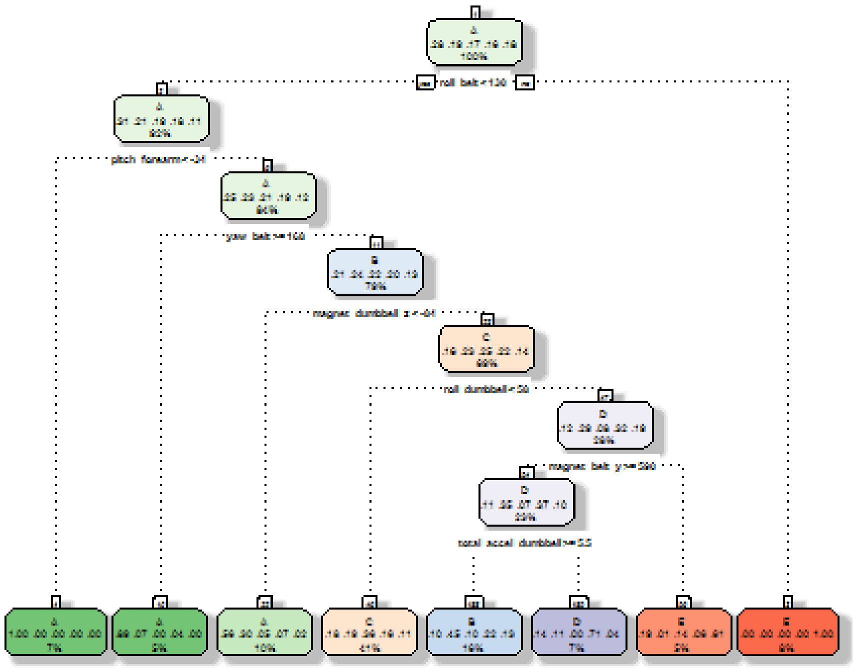
```
# Train on training set 1 of 4 with no extra features.
set.seed(666)
modFit <- train(df_small_training1$classe ~ ., data = df_small_training1,
method="rpart")
print(modFit, digits=3)
```

```
## 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.
```

```r
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) *
```

```r
fancyRpartPlot(modFit$finalModel)
```



Rattle 2015-Aug-23 01:57:34 Viji

```
# 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
```

low accuracy rate (0.5584)

```r
# 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.
```

```r
# 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

First I decided to assess the impact/value of including preprocessing.

```r
# 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  0.00291      0.00367
```

```
##   27    0.957     0.945  0.00740      0.00937
##   52    0.951     0.939  0.01226      0.01549
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

```r
# 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 556  10   1   0   0
##          B   2 360  13   0   2
##          C   0   9 323   4   5
##          D   0   1   5 313   2
##          E   0   0   0   4 351
##
## Overall Statistics
##
##                Accuracy : 0.9704
##                  95% CI : (0.9619, 0.9775)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9626
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9964   0.9474   0.9444   0.9751   0.9750
## Specificity            0.9922   0.9892   0.9889   0.9951   0.9975
## Pos Pred Value         0.9806   0.9549   0.9472   0.9751   0.9887
## Neg Pred Value         0.9986   0.9874   0.9883   0.9951   0.9944
## Prevalence             0.2845   0.1938   0.1744   0.1637   0.1836
## Detection Rate         0.2835   0.1836   0.1647   0.1596   0.1790
## Detection Prevalence   0.2891   0.1922   0.1739   0.1637   0.1810
## Balanced Accuracy      0.9943   0.9683   0.9667   0.9851   0.9863
```

```r
# Run against 20 testing set
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
```

```r
# Train on training set 1 of 4 with only both preprocessing and cross
validation.
set.seed(666)
```

```r
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  0.00172      0.00217
##   27    0.955     0.943  0.00588      0.00743
##   52    0.952     0.939  0.01061      0.01341
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

```r
# 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 556  12   1   0   1
##          B   2 358  12   0   0
##          C   0   9 324   6   5
##          D   0   1   5 310   2
##          E   0   0   0   5 352
##
## 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            0.9964   0.9421   0.9474   0.9657   0.9778
## Specificity            0.9900   0.9911   0.9876   0.9951   0.9969
## Pos Pred Value         0.9754   0.9624   0.9419   0.9748   0.9860
## Neg Pred Value         0.9986   0.9862   0.9889   0.9933   0.9950
## Prevalence             0.2845   0.1938   0.1744   0.1637   0.1836
## Detection Rate         0.2835   0.1826   0.1652   0.1581   0.1795
## Detection Prevalence   0.2907   0.1897   0.1754   0.1622   0.1820
## Balanced Accuracy      0.9932   0.9666   0.9675   0.9804   0.9873
```

```
# Run against 20 testing
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    0.953     0.941  0.00953      0.01210
##   27    0.952     0.939  0.00699      0.00889
##   52    0.941     0.926  0.00539      0.00683
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```
# 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 547  12   0   3   0
##          B   2 351  22   0   1
##          C   0  12 314  19   6
##          D   2   1   2 293   6
##          E   1   0   0   3 344
##
## Overall Statistics
##
##                Accuracy : 0.9526
##                  95% CI : (0.9422, 0.9616)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.94
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9909   0.9335   0.9290   0.9214   0.9636
## Specificity            0.9892   0.9840   0.9769   0.9932   0.9975
## Pos Pred Value         0.9733   0.9335   0.8946   0.9638   0.9885
## Neg Pred Value         0.9964   0.9840   0.9849   0.9847   0.9918
## Prevalence             0.2844   0.1937   0.1741   0.1638   0.1839
## Detection Rate         0.2818   0.1808   0.1618   0.1510   0.1772
## Detection Prevalence   0.2895   0.1937   0.1808   0.1566   0.1793
## Balanced Accuracy      0.9901   0.9588   0.9530   0.9573   0.9805
```

```
# Run against 20 testing set
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     0.936  0.00572      0.00724
##   27    0.949     0.936  0.00953      0.01206
##   52    0.942     0.926  0.01020      0.01293
##
## 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 555  10   0   1   0
##          B   1 358  18   0   3
##          C   1  12 320   7   3
##          D   2   1   3 313   1
##          E   1   0   3   2 355
##
## Overall Statistics
##
##                Accuracy : 0.965
##                  95% CI : (0.9559, 0.9726)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9557
##  Mcnemar's Test P-Value : 0.0782
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9911   0.9396   0.9302   0.9690   0.9807
## Specificity            0.9922   0.9862   0.9859   0.9957   0.9963
## Pos Pred Value         0.9806   0.9421   0.9329   0.9781   0.9834
## Neg Pred Value         0.9964   0.9855   0.9852   0.9939   0.9956
## Prevalence             0.2843   0.1934   0.1746   0.1640   0.1838
## Detection Rate         0.2817   0.1817   0.1624   0.1589   0.1802
## Detection Prevalence   0.2873   0.1929   0.1741   0.1624   0.1832
## Balanced Accuracy      0.9916   0.9629   0.9580   0.9824   0.9885
```

```
# Run against 20 testing set
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    0.947     0.933  0.00969      0.01228
##   27    0.957     0.945  0.00722      0.00914
##   52    0.947     0.933  0.01031      0.01307
##
## 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 552  19   0   0   0
##          B   5 358  20   3   2
##          C   2   4 315   8   7
##          D   1   0   8 311   6
##          E   0   0   0   1 347
##
## 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            0.9857   0.9396   0.9184   0.9628   0.9586
## Specificity            0.9865   0.9811   0.9871   0.9909   0.9994
## Pos Pred Value         0.9667   0.9227   0.9375   0.9540   0.9971
## Neg Pred Value         0.9943   0.9855   0.9829   0.9927   0.9907
## Prevalence             0.2844   0.1935   0.1742   0.1640   0.1838
## Detection Rate         0.2803   0.1818   0.1600   0.1579   0.1762
## Detection Prevalence   0.2900   0.1971   0.1706   0.1656   0.1767
## Balanced Accuracy      0.9861   0.9604   0.9527   0.9769   0.9790
```

```
# 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
```

CONCLUSION

I received three separate predictions by appling the 4 models against the actual 20 item training set:

A) 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

B) 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

C) 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%.