Practical Machine Learning Course Project

Melanie Carlson

Sunday, June 21, 2015

The goal of this project is to determine how the particpant did the excercise. This report will answer the following: How the model was built, How it was cross validated, Expected Sample Error and why I made the choices I did.

Question

Use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants who were asked to perform barbell lifts correctly and incorrectly in 5 different ways to create an alogorithm that correctly identifies the activity quality. 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

Input Data

Load necessary libraries

```
library(AppliedPredictiveModeling)

## Warning: package 'AppliedPredictiveModeling' was built under R version
## 3.2.1

library(caret)

## Warning: package 'caret' was built under R version 3.2.1
## Loading required package: lattice
## Loading required package: ggplot2

library(rattle)

## Warning: package 'rattle' was built under R version 3.2.1
## 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)
```

```
## Warning: package 'randomForest' was built under R version 3.2.1
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
library(knitr)
## Warning: package 'knitr' was built under R version 3.2.1
library(e1071)
## Warning: package 'e1071' was built under R version 3.2.1
```

Upload Data

```
setwd("C:/Users/Melanie/Desktop/R Code Class/Machine Learning")
df_training <- read.csv("pml-training.csv", na.strings=c("NA",""),
header=TRUE)
colnames_train <- colnames(df_training)
df_testing <- read.csv("pml-testing.csv", 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

Remove Incomplete Data

```
# 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)
```

```r
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.

```r
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"
```

```r
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 Covariates for Variatability

```r
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
```

All False, so no need to remove any covariates due to lack of variatability

Algorithm

Devided the data up into 4 sets, so able to do multiple trials of the algorithm to avoid overfitting the predictor and so things would run quicker.

```
set.seed(2)
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)
```

```r
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

Used classification trees “out of the box” and then introduce preprocessing and cross validation.

Evaluation

Classification Tree

```r
set.seed(2)
##Train
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
##   0.0356  0.517     0.3773  0.0685       0.1054
##   0.0596  0.375     0.1454  0.0399       0.0645
##   0.1157  0.340     0.0907  0.0373       0.0523
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.0356.
```

```
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 2696 1860 A (0.31 0.21 0.19 0.18 0.11)
##      4) pitch_forearm< -34 227    1 A (1 0.0044 0 0 0) *
##      5) pitch_forearm>=-34 2469 1860 A (0.25 0.23 0.21 0.2 0.12)
##       10) magnet_dumbbell_y< 426 2051 1460 A (0.29 0.18 0.24 0.19 0.099)
##         20) roll_forearm< 122 1287  752 A (0.42 0.17 0.19 0.16 0.057) *
##         21) roll_forearm>=122 764  513 C (0.075 0.19 0.33 0.24 0.17) *
##       11) magnet_dumbbell_y>=426 418  219 B (0.038 0.48 0.038 0.23 0.22) *
##    3) roll_belt>=130 250    3 E (0.012 0 0 0 0.99) *
```

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

```
##Run Against my Test Set
```

```r
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 515 154 146 133  44
##          B   7 143  15  59  54
##          C  32  83 181 129  97
##          D   0   0   0   0   0
##          E   4   0   0   0 165
##
## Overall Statistics
##
##                Accuracy : 0.512
##                  95% CI : (0.4896, 0.5343)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3632
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9229  0.37632   0.5292   0.0000  0.45833
## Specificity            0.6600  0.91461   0.7894   1.0000  0.99750
## Pos Pred Value         0.5192  0.51439   0.3467      NaN  0.97633
## Neg Pred Value         0.9556  0.85918   0.8881   0.8363  0.89118
## Prevalence             0.2845  0.19378   0.1744   0.1637  0.18358
## Detection Rate         0.2626  0.07292   0.0923   0.0000  0.08414
## Detection Prevalence   0.5059  0.14176   0.2662   0.0000  0.08618
## Balanced Accuracy      0.7915  0.64546   0.6593   0.5000  0.72792
```

Low Accuracy, so attempted again using reprocessing and cross validation

```
set.seed(2)
##Train
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: 2209, 2211, 2209, 2209
##
## Resampling results across tuning parameters:
##
##   cp      Accuracy  Kappa   Accuracy SD  Kappa SD
##   0.0356  0.525     0.3954  0.01009      0.0138
##   0.0596  0.367     0.1265  0.00378      0.0054
##   0.1157  0.345     0.0932  0.04061      0.0622
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.0356.
```

```
##Run Against my Test Set
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 515 154 146 133  44
##          B   7 143  15  59  54
##          C  32  83 181 129  97
##          D   0   0   0   0   0
##          E   4   0   0   0 165
##
## Overall Statistics
##
##                Accuracy : 0.512
##                  95% CI : (0.4896, 0.5343)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3632
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9229  0.37632   0.5292   0.0000  0.45833
## Specificity            0.6600  0.91461   0.7894   1.0000  0.99750
## Pos Pred Value         0.5192  0.51439   0.3467      NaN  0.97633
## Neg Pred Value         0.9556  0.85918   0.8881   0.8363  0.89118
## Prevalence             0.2845  0.19378   0.1744   0.1637  0.18358
## Detection Rate         0.2626  0.07292   0.0923   0.0000  0.08414
## Detection Prevalence   0.5059  0.14176   0.2662   0.0000  0.08618
## Balanced Accuracy      0.7915  0.64546   0.6593   0.5000  0.72792
```

Little improvement on accurancy, so decided to use Random Forest.

Random Forest

```
set.seed(2)
##Train
```

```r
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: 2209, 2211, 2209, 2209
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
##    2    0.946     0.931  0.01101      0.01395
##   27    0.952     0.940  0.00573      0.00725
##   52    0.946     0.932  0.00806      0.01022
##
## 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 my Test Set
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 553   8   0   2   0
##          B   2 361  18   2   9
##          C   2  10 320  14   1
##          D   0   1   4 301   1
```

```
##          E   1   0   0   2 349
##
## Overall Statistics
##
##                Accuracy : 0.9607
##                  95% CI : (0.9512, 0.9689)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9503
##  Mcnemar's Test P-Value : 0.002504
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9910   0.9500   0.9357   0.9377   0.9694
## Specificity            0.9929   0.9804   0.9833   0.9963   0.9981
## Pos Pred Value         0.9822   0.9209   0.9222   0.9805   0.9915
## Neg Pred Value         0.9964   0.9879   0.9864   0.9879   0.9932
## Prevalence             0.2845   0.1938   0.1744   0.1637   0.1836
## Detection Rate         0.2820   0.1841   0.1632   0.1535   0.1780
## Detection Prevalence   0.2871   0.1999   0.1770   0.1566   0.1795
## Balanced Accuracy      0.9920   0.9652   0.9595   0.9670   0.9838
```

```
##Ran Against Course Provided Test Set
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
```

attempted again using reprocessing and cross validation

```
set.seed(2)
##Train
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: 2209, 2211, 2209, 2209
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
##    2    0.946     0.931  0.01276      0.01616
##   27    0.951     0.938  0.00588      0.00744
##   52    0.946     0.932  0.00798      0.01014
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

```
##Run Against my Test Set
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 553   7   0   2   0
##          B   3 362  17   2   7
##          C   2  10 321  12   1
##          D   0   1   3 303   1
##          E   0   0   1   2 351
##
## Overall Statistics
```

```
##
##                Accuracy : 0.9638
##                  95% CI : (0.9545, 0.9716)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9542
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9910   0.9526   0.9386   0.9439   0.9750
## Specificity            0.9936   0.9817   0.9846   0.9970   0.9981
## Pos Pred Value         0.9840   0.9258   0.9277   0.9838   0.9915
## Neg Pred Value         0.9964   0.9885   0.9870   0.9891   0.9944
## Prevalence             0.2845   0.1938   0.1744   0.1637   0.1836
## Detection Rate         0.2820   0.1846   0.1637   0.1545   0.1790
## Detection Prevalence   0.2866   0.1994   0.1764   0.1571   0.1805
## Balanced Accuracy      0.9923   0.9671   0.9616   0.9704   0.9866
```

```r
##Ran Against Course Provided Test 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
```

Mixed results to decided to run again using my second set of data with only Cross Validation

```r
set.seed(2)
##Train
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, 2188, 2187
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
##    2    0.948     0.934  0.00783      0.00991
##   27    0.955     0.943  0.00847      0.01069
##   52    0.948     0.934  0.00605      0.00760
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

```
##Run Against my Test Set
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 549  10   0   0   0
##          B   2 363   8   1   0
##          C   0   3 327  18   3
##          D   0   0   3 298   6
##          E   1   0   0   1 348
##
## Overall Statistics
##
##                Accuracy : 0.9711
```

```
##                    95% CI : (0.9627, 0.9781)
##       No Information Rate : 0.2844
##       P-Value [Acc > NIR] : < 2.2e-16
##
##                     Kappa : 0.9635
##    Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9946   0.9654   0.9675   0.9371   0.9748
## Specificity            0.9928   0.9930   0.9850   0.9945   0.9987
## Pos Pred Value         0.9821   0.9706   0.9316   0.9707   0.9943
## Neg Pred Value         0.9978   0.9917   0.9931   0.9878   0.9943
## Prevalence             0.2844   0.1937   0.1741   0.1638   0.1839
## Detection Rate         0.2828   0.1870   0.1685   0.1535   0.1793
## Detection Prevalence   0.2880   0.1927   0.1808   0.1582   0.1803
## Balanced Accuracy      0.9937   0.9792   0.9762   0.9658   0.9868
```

```
##Ran Against Course Provided Test 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
```

Looked good, so ran my 3rd set of data with only Cross Validation

```
set.seed(2)
##Train
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: 2220, 2220, 2220, 2220
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
##    2    0.951     0.938  0.00943      0.0120
##   27    0.948     0.934  0.01547      0.0197
##   52    0.942     0.927  0.01777      0.0225
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```r
##Run Against my Test Set
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 558  16   0   1   1
##          B   1 357  23   0   1
##          C   0   7 319  24   5
##          D   1   1   2 298   3
##          E   0   0   0   0 352
##
## Overall Statistics
##
##                  Accuracy : 0.9563
##                    95% CI : (0.9464, 0.9649)
##       No Information Rate : 0.2843
##       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            0.9964   0.9370   0.9273   0.9226   0.9724
## Specificity            0.9872   0.9843   0.9779   0.9957   1.0000
## Pos Pred Value         0.9688   0.9346   0.8986   0.9770   1.0000
## Neg Pred Value         0.9986   0.9849   0.9845   0.9850   0.9938
## Prevalence             0.2843   0.1934   0.1746   0.1640   0.1838
## Detection Rate         0.2832   0.1812   0.1619   0.1513   0.1787
## Detection Prevalence   0.2924   0.1939   0.1802   0.1548   0.1787
## Balanced Accuracy      0.9918   0.9606   0.9526   0.9592   0.9862
```

```
##Ran Against Course Provided Test 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
```

Looked good, so ran my final set of data with only Cross Validation

```
set.seed(2)
##Train
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: 2217, 2219, 2219, 2219
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
##    2    0.949     0.936  0.00633      0.00805
##   27    0.955     0.943  0.00697      0.00886
##   52    0.950     0.937  0.00403      0.00514
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

```
##Run Against my Test Set
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 556  20   2   0   0
##          B   3 341   8   0   3
##          C   0  19 329   4   3
##          D   1   1   4 318   4
##          E   0   0   0   1 352
##
## Overall Statistics
##
##                Accuracy : 0.9629
##                  95% CI : (0.9536, 0.9708)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9531
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9929   0.8950   0.9592   0.9845   0.9724
## Specificity            0.9844   0.9912   0.9840   0.9939   0.9994
## Pos Pred Value         0.9619   0.9606   0.9268   0.9695   0.9972
## Neg Pred Value         0.9971   0.9752   0.9913   0.9970   0.9938
## Prevalence             0.2844   0.1935   0.1742   0.1640   0.1838
## Detection Rate         0.2824   0.1732   0.1671   0.1615   0.1788
## Detection Prevalence   0.2936   0.1803   0.1803   0.1666   0.1793
## Balanced Accuracy      0.9886   0.9431   0.9716   0.9892   0.9859
```

```
##Ran Against Course Provided Test 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
```

Out of Sample Error Rate

Average of the sample error rates derived by applying the random forest method with both preprocessing and cross validation against test sets 1-4 yielding a predicted out of sample rate of 0.03584.