Practical ML Project - Classifying Manner of Exercises

Michael Nichols

March 23, 2018

Executive Summary

The goal of the project is to predict the manner in which individuals did a certain exercise. In this project, the goal is to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. ### Target Variable The outcome variable is classe, a factor variable with 5 levels. For this data set, "participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in 5 different fashions:

- 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)
- 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."

Prediction evaluations will be based on maximizing the accuracy and minimizing the out-of-sample error. All other available variables after cleaning will be used for prediction.

Findings

Basic Steps to Build the Model:

0. "Set the seed." Ensure reproducibility for the model.

1. Read in the dataset

2A. Preserve only the variables that included data in at least 98% of records. This removed 100 variables. (160 to 60)

2B. Remove 7 additional variables that did not appear to have any predictive value. This included things like name and times of the records.

3. Partition the dataset. The original data came in two CSV files, one labeled 'training' and one labeled 'testing.' The test dataset included the final 20 questions for the final evaluation. In order to build the model without overfitting, I split the training set randomly in a 70/30 split into a training subset and validation set. This validation is used to test the accuracy of the models.

4. Build & train the models! By attempting to predict the classe, the proper algorithm to use for the model needed to work with factor variables. (i.e. a classification problem) With this in mind, I built two models and compared the prediction accuracy of each to decide the best fit.

4A. Attempt 1 - Random Forest. Out-of-the-box random forest model from the randomForest R package produced the greatest accuracy.

4B. Attempt 2 - Generalized Boosted Model. Despite using custom tuning parameters (3-fold cross validation), this model did not perform as well as the random forest model. This model's accuracy was 96.18%.

5. Test each model against the validation set and examine the accuracy.

6. Predict using the best model (Random Forest) against the test set.

Prediction Results using a Random Forest model:

Accuracy: 99.42% Out of Sample Error Rate: .58%

Problem 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://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Environment Setup: load dataset, R packages, & ensure reproducibility

```r
# set working directory
setwd("S:/Documents/R")

# load R packages
library(data.table)  # for quick file reading
library(caret)  # for model building - ideal for classification & regression problems
library(randomForest)  # for efficient random forest model building

set.seed(16)  # for reproducibility

# download train/test datasets for evaluation
training <- fread("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",
    na.strings = c("NA", "#DIV/0!", ""))

# test dataset to be used for final evaluation only & will be
# ignored for all model building
testing <- fread("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",
    na.strings = c("NA", "#DIV/0!", ""))
```

Initial Exploration & Cleansing

```r
# evaluate datasets
dim(training)
```

```
## [1] 19622   160
```

```r
dim(testing)
```

```
## [1]  20 160
```

```r
# as expected, test dataset contains 20 records for final
# evaluation
```

```
# convert to DF's
training <- as.data.frame(training)
testing <- as.data.frame(testing)

# Remove columns with >= 98% of NA or '' values
relevant_Columns <- !apply(training, 2, function(x) sum(is.na(x)) >
    0.98 || sum(x == "") > 0.98)
training <- training[, relevant_Columns]
testing <- testing[, relevant_Columns]

# convert target variable to factor
training$classe <- as.factor(training$classe)
# include classe in testing set as NA
testing$classe <- NA

# review training and test sets again to remove additional
# unneeded columns names(testing) # first seven variables can
# be removed based on our goal (times, username, etc.)
# names(training)
testing[, 1:7] <- NULL
training[, 1:7] <- NULL

# show final dataset dimensions ~ over 100 unneccessary
# variables removed
dim(training)
```

```
## [1] 19622    53
```

```
dim(testing)
```

```
## [1] 20 54
```

Prepare cross validation

Since we already have the final test set, we'll create a validation set within the training dataset using a 70/30 random split.

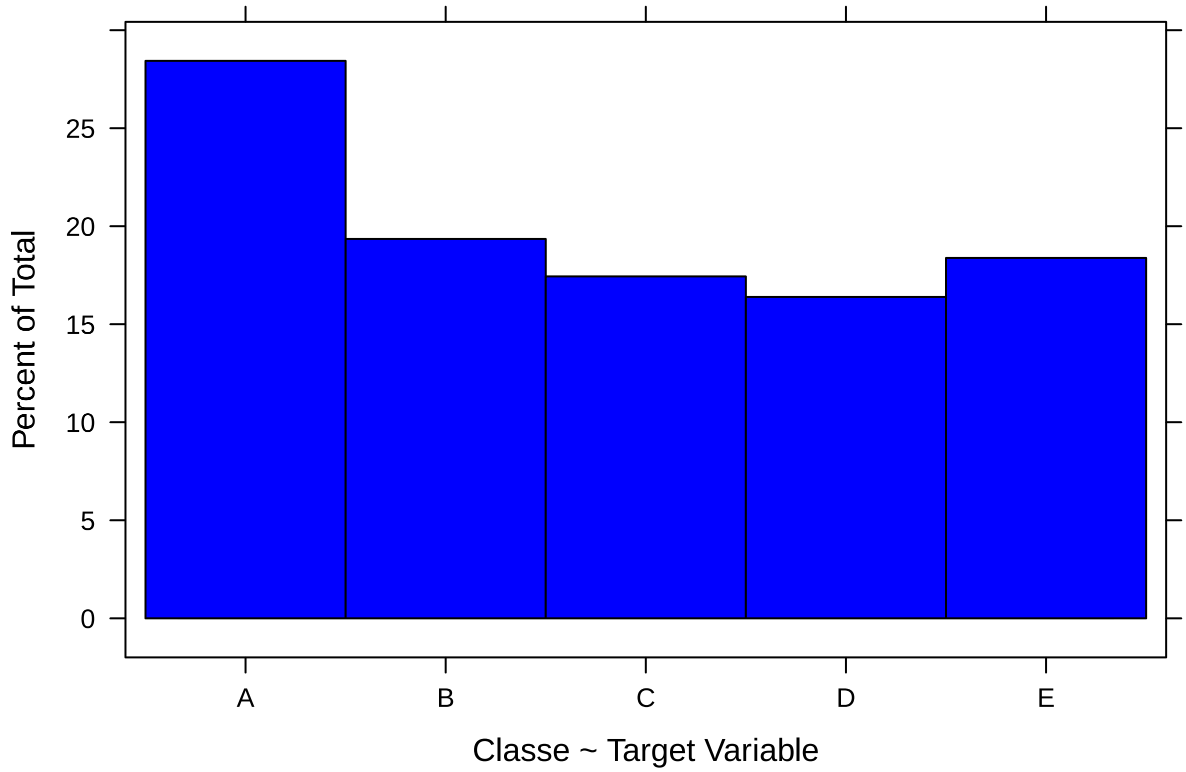
```
inValidation <- createDataPartition(y = training$classe, p = 0.7,
    list = FALSE)
TrainingSet <- training[inValidation, ]
ValidationSet <- training[-inValidation, ]

dim(TrainingSet)
```

```
## [1] 13737    53
```

```
dim(ValidationSet)
```

```
## [1] 5885   53
```




```
##    A    B    C    D    E
## 3906 2658 2396 2252 2525
```

Algorith Attempt #1 –> Random Forest

```
# Algorithm 1: Random Forest build the model
modFit_RF <- randomForest(classe ~ ., data = TrainingSet, ntree = 100,
    mtry = 16)

# predict using decision tree model against validation set
prediction_RF <- predict(modFit_RF, ValidationSet)

# review accuracy against the validation set
confusionMatrix(prediction_RF, ValidationSet$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1672    6    0    0    0
##          B    2 1131    3    1    0
##          C    0    2 1021    6    3
##          D    0    0    2  956    2
##          E    0    0    0    1 1077
##
```

```
## Overall Statistics
##
##                Accuracy : 0.9952
##                  95% CI : (0.9931, 0.9968)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.994
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9988   0.9930   0.9951   0.9917   0.9954
## Specificity            0.9986   0.9987   0.9977   0.9992   0.9998
## Pos Pred Value         0.9964   0.9947   0.9893   0.9958   0.9991
## Neg Pred Value         0.9995   0.9983   0.9990   0.9984   0.9990
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2841   0.1922   0.1735   0.1624   0.1830
## Detection Prevalence   0.2851   0.1932   0.1754   0.1631   0.1832
## Balanced Accuracy      0.9987   0.9959   0.9964   0.9954   0.9976
```

Algorithm Attempt #2 –> Generalized Boosted Model

```r
# Algorithm 2: Generalized Boosted Model Tune the model:
# utilize 3-fold cross validation to further break apart the
# training set and determine best model settings with GBM,
# the 3 tuning parameters checked will be trees, shrinkage,
# and interaction depth.
objControl <- trainControl(method = "cv", number = 3, returnResamp = "none",
    classProbs = TRUE)

# train the model
modFit_GBM <- train(classe ~ ., data = TrainingSet, method = "gbm",
    trControl = objControl, metric = "Accuracy", preProc = c("center",
        "scale"))
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1271
##      2        1.5229             nan     0.1000    0.0876
##      3        1.4642             nan     0.1000    0.0651
##      4        1.4195             nan     0.1000    0.0537
##      5        1.3840             nan     0.1000    0.0449
##      6        1.3544             nan     0.1000    0.0448
##      7        1.3253             nan     0.1000    0.0368
##      8        1.2994             nan     0.1000    0.0302
##      9        1.2795             nan     0.1000    0.0326
##     10        1.2564             nan     0.1000    0.0303
##     20        1.0996             nan     0.1000    0.0165
##     40        0.9298             nan     0.1000    0.0086
##     60        0.8240             nan     0.1000    0.0063
##     80        0.7420             nan     0.1000    0.0045
##    100        0.6796             nan     0.1000    0.0036
```

```
##    120        0.6252             nan     0.1000    0.0027
##    140        0.5816             nan     0.1000    0.0020
##    150        0.5612             nan     0.1000    0.0012
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1868
##      2        1.4895             nan     0.1000    0.1316
##      3        1.4056             nan     0.1000    0.1042
##      4        1.3384             nan     0.1000    0.0873
##      5        1.2820             nan     0.1000    0.0736
##      6        1.2349             nan     0.1000    0.0656
##      7        1.1927             nan     0.1000    0.0593
##      8        1.1545             nan     0.1000    0.0463
##      9        1.1236             nan     0.1000    0.0428
##     10        1.0952             nan     0.1000    0.0466
##     20        0.8930             nan     0.1000    0.0180
##     40        0.6762             nan     0.1000    0.0095
##     60        0.5496             nan     0.1000    0.0056
##     80        0.4617             nan     0.1000    0.0044
##    100        0.3961             nan     0.1000    0.0039
##    120        0.3454             nan     0.1000    0.0021
##    140        0.3065             nan     0.1000    0.0024
##    150        0.2872             nan     0.1000    0.0022
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2371
##      2        1.4595             nan     0.1000    0.1582
##      3        1.3578             nan     0.1000    0.1218
##      4        1.2789             nan     0.1000    0.1011
##      5        1.2142             nan     0.1000    0.0788
##      6        1.1625             nan     0.1000    0.0855
##      7        1.1085             nan     0.1000    0.0738
##      8        1.0629             nan     0.1000    0.0530
##      9        1.0277             nan     0.1000    0.0630
##     10        0.9884             nan     0.1000    0.0523
##     20        0.7543             nan     0.1000    0.0231
##     40        0.5284             nan     0.1000    0.0125
##     60        0.4053             nan     0.1000    0.0090
##     80        0.3193             nan     0.1000    0.0043
##    100        0.2612             nan     0.1000    0.0026
##    120        0.2166             nan     0.1000    0.0029
##    140        0.1830             nan     0.1000    0.0016
##    150        0.1699             nan     0.1000    0.0014
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1257
##      2        1.5242             nan     0.1000    0.0825
##      3        1.4683             nan     0.1000    0.0659
##      4        1.4242             nan     0.1000    0.0568
##      5        1.3888             nan     0.1000    0.0436
##      6        1.3597             nan     0.1000    0.0430
##      7        1.3315             nan     0.1000    0.0407
##      8        1.3046             nan     0.1000    0.0350
##      9        1.2803             nan     0.1000    0.0329
```

```
##     10        1.2587             nan     0.1000    0.0283
##     20        1.1049             nan     0.1000    0.0183
##     40        0.9335             nan     0.1000    0.0086
##     60        0.8254             nan     0.1000    0.0058
##     80        0.7450             nan     0.1000    0.0050
##    100        0.6818             nan     0.1000    0.0032
##    120        0.6298             nan     0.1000    0.0031
##    140        0.5852             nan     0.1000    0.0024
##    150        0.5664             nan     0.1000    0.0017
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1880
##      2        1.4871             nan     0.1000    0.1307
##      3        1.4046             nan     0.1000    0.1016
##      4        1.3384             nan     0.1000    0.0824
##      5        1.2836             nan     0.1000    0.0704
##      6        1.2396             nan     0.1000    0.0590
##      7        1.2006             nan     0.1000    0.0542
##      8        1.1650             nan     0.1000    0.0603
##      9        1.1273             nan     0.1000    0.0489
##     10        1.0959             nan     0.1000    0.0403
##     20        0.8989             nan     0.1000    0.0267
##     40        0.6828             nan     0.1000    0.0088
##     60        0.5539             nan     0.1000    0.0068
##     80        0.4640             nan     0.1000    0.0051
##    100        0.3960             nan     0.1000    0.0035
##    120        0.3461             nan     0.1000    0.0022
##    140        0.3039             nan     0.1000    0.0014
##    150        0.2870             nan     0.1000    0.0010
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2302
##      2        1.4612             nan     0.1000    0.1544
##      3        1.3599             nan     0.1000    0.1257
##      4        1.2791             nan     0.1000    0.1047
##      5        1.2125             nan     0.1000    0.0839
##      6        1.1591             nan     0.1000    0.0778
##      7        1.1099             nan     0.1000    0.0737
##      8        1.0633             nan     0.1000    0.0608
##      9        1.0231             nan     0.1000    0.0527
##     10        0.9895             nan     0.1000    0.0554
##     20        0.7613             nan     0.1000    0.0221
##     40        0.5288             nan     0.1000    0.0140
##     60        0.4008             nan     0.1000    0.0065
##     80        0.3214             nan     0.1000    0.0049
##    100        0.2650             nan     0.1000    0.0039
##    120        0.2181             nan     0.1000    0.0020
##    140        0.1865             nan     0.1000    0.0020
##    150        0.1729             nan     0.1000    0.0018
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1274
##      2        1.5233             nan     0.1000    0.0861
##      3        1.4660             nan     0.1000    0.0682
```

```
##      4        1.4220             nan     0.1000    0.0516
##      5        1.3882             nan     0.1000    0.0484
##      6        1.3556             nan     0.1000    0.0444
##      7        1.3266             nan     0.1000    0.0341
##      8        1.3043             nan     0.1000    0.0345
##      9        1.2815             nan     0.1000    0.0338
##     10        1.2593             nan     0.1000    0.0278
##     20        1.1057             nan     0.1000    0.0193
##     40        0.9313             nan     0.1000    0.0080
##     60        0.8227             nan     0.1000    0.0048
##     80        0.7455             nan     0.1000    0.0052
##    100        0.6803             nan     0.1000    0.0019
##    120        0.6275             nan     0.1000    0.0031
##    140        0.5824             nan     0.1000    0.0022
##    150        0.5645             nan     0.1000    0.0024
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1858
##      2        1.4890             nan     0.1000    0.1290
##      3        1.4064             nan     0.1000    0.0994
##      4        1.3408             nan     0.1000    0.0845
##      5        1.2872             nan     0.1000    0.0794
##      6        1.2374             nan     0.1000    0.0600
##      7        1.1982             nan     0.1000    0.0519
##      8        1.1647             nan     0.1000    0.0542
##      9        1.1301             nan     0.1000    0.0475
##     10        1.0996             nan     0.1000    0.0450
##     20        0.9015             nan     0.1000    0.0183
##     40        0.6870             nan     0.1000    0.0117
##     60        0.5590             nan     0.1000    0.0082
##     80        0.4627             nan     0.1000    0.0041
##    100        0.3994             nan     0.1000    0.0023
##    120        0.3449             nan     0.1000    0.0031
##    140        0.3028             nan     0.1000    0.0028
##    150        0.2855             nan     0.1000    0.0019
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2343
##      2        1.4589             nan     0.1000    0.1643
##      3        1.3554             nan     0.1000    0.1180
##      4        1.2780             nan     0.1000    0.1073
##      5        1.2121             nan     0.1000    0.0818
##      6        1.1589             nan     0.1000    0.0752
##      7        1.1113             nan     0.1000    0.0665
##      8        1.0673             nan     0.1000    0.0591
##      9        1.0290             nan     0.1000    0.0607
##     10        0.9897             nan     0.1000    0.0445
##     20        0.7540             nan     0.1000    0.0212
##     40        0.5328             nan     0.1000    0.0072
##     60        0.4001             nan     0.1000    0.0064
##     80        0.3185             nan     0.1000    0.0028
##    100        0.2607             nan     0.1000    0.0021
##    120        0.2196             nan     0.1000    0.0023
##    140        0.1856             nan     0.1000    0.0013
```

```
##    150        0.1728             nan     0.1000    0.0012
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2384
##      2        1.4595             nan     0.1000    0.1584
##      3        1.3589             nan     0.1000    0.1232
##      4        1.2808             nan     0.1000    0.1025
##      5        1.2159             nan     0.1000    0.0937
##      6        1.1581             nan     0.1000    0.0746
##      7        1.1108             nan     0.1000    0.0669
##      8        1.0691             nan     0.1000    0.0576
##      9        1.0323             nan     0.1000    0.0624
##     10        0.9944             nan     0.1000    0.0610
##     20        0.7564             nan     0.1000    0.0223
##     40        0.5329             nan     0.1000    0.0098
##     60        0.4051             nan     0.1000    0.0066
##     80        0.3269             nan     0.1000    0.0039
##    100        0.2714             nan     0.1000    0.0031
##    120        0.2261             nan     0.1000    0.0017
##    140        0.1931             nan     0.1000    0.0025
##    150        0.1780             nan     0.1000    0.0008
```

```r
# predict using GBM on the validation set
prediction_GBM <- predict(modFit_GBM, ValidationSet)

# review accuracy against the validation set
confusionMatrix(prediction_GBM, ValidationSet$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1652   36    0    2    3
##          B   17 1065   18    5   12
##          C    1   36  997   34    6
##          D    3    2    9  914   13
##          E    1    0    2    9 1048
##
## Overall Statistics
##
##                Accuracy : 0.9645
##                  95% CI : (0.9594, 0.9691)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9551
##  Mcnemar's Test P-Value : 1.723e-06
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9869   0.9350   0.9717   0.9481   0.9686
## Specificity            0.9903   0.9890   0.9842   0.9945   0.9975
## Pos Pred Value         0.9758   0.9534   0.9283   0.9713   0.9887
## Neg Pred Value         0.9948   0.9845   0.9940   0.9899   0.9930
```

```
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2807   0.1810   0.1694   0.1553   0.1781
## Detection Prevalence   0.2877   0.1898   0.1825   0.1599   0.1801
## Balanced Accuracy      0.9886   0.9620   0.9779   0.9713   0.9830
```

Both models perform extremely well, but across the board, the random forest model outperforms the generalized boosted model. The final accuracy of the RF model against the validation set exceeds 99%.

```
##      Accuracy     Kappa AccuracyLower AccuracyUpper AccuracyNull
## RF  0.9952421 0.9939814     0.9931309     0.9968362     0.284452
## GBM 0.9644860 0.9550631     0.9594365     0.9690679     0.284452
##     AccuracyPValue McnemarPValue
## RF               0           NaN
## GBM              0  1.723477e-06
```

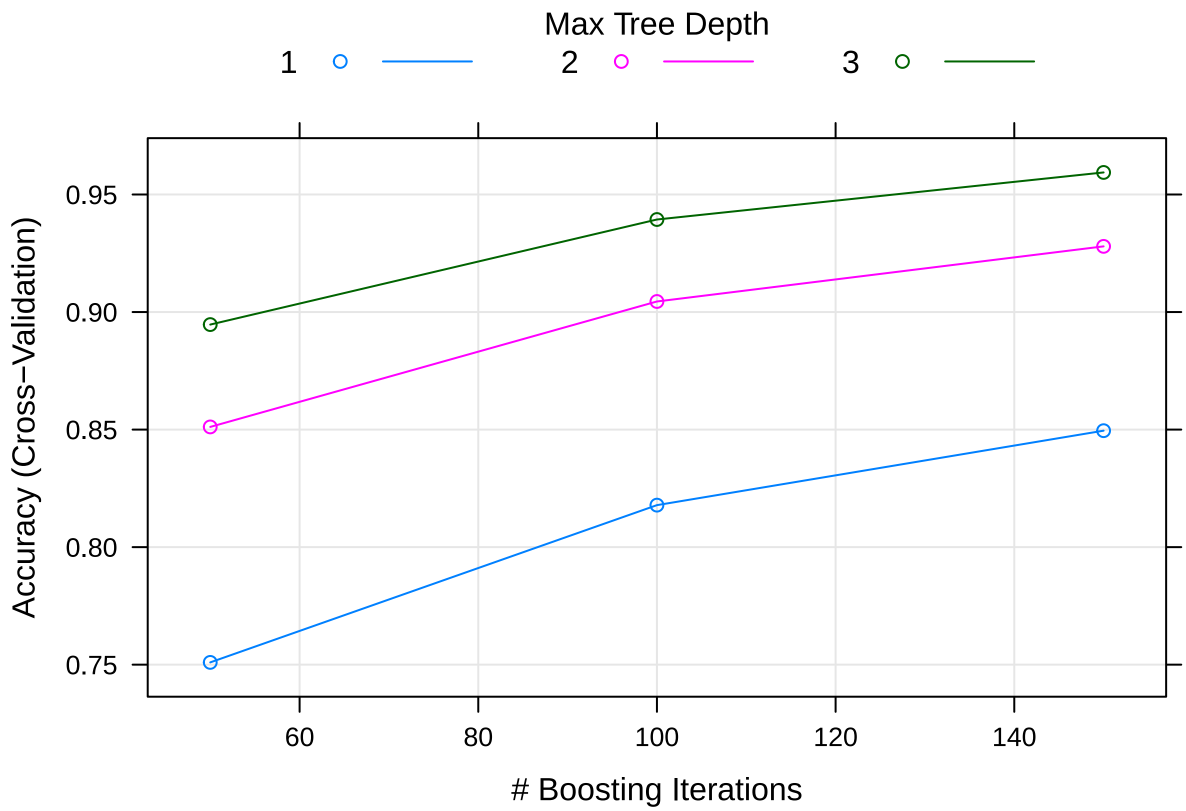
Final Prediction / Quiz

```
predict(modFit_RF, testing)
```

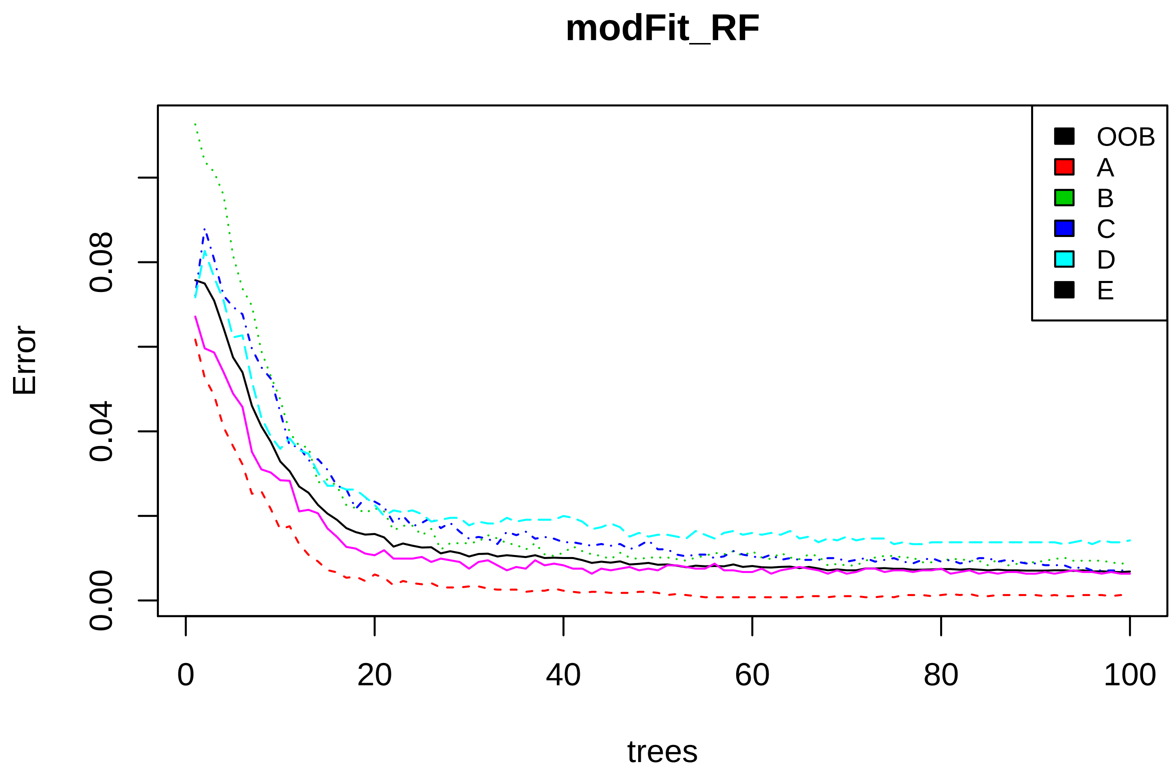
```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  B  A  B  A  A  E  D  B  A  A  B  C  B  A  E  E  A  B  B  B
## Levels: A B C D E
```

Appendix

Plot 1: displays model accuracy increases across cross validation sets as boosting iterations rise



Plot 2: displays decreasing model error rate as # of trees expands



Plot 3: displays top 10 most important variables based on predictions for each outcome variable option