# Coursera Machine Learning Project

Ethan Schnelle

October 23, 2019

## Human Activity Recognition

### 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, the goal is to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. These participants were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information on the Human Activity Study 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)*

### Data

**Training Data** *(https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv)*

**Test Data** *(https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)*

**Data Source** *(http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har)*

### Project Goal

The goal of this project is to predict the manner in which participants did their exercise. This is the **"classe"** variable in the **Training** set. Any other variables are predictor factor candidates. This report describes our model was built, how cross validation was used, and provides my thoughts on the expected out of sample error, and why I made the choices I did. After choosing a model, it will be used to predict on the 20 different cases in the **Test** set.

### Background for Work

**Reproduceability** To reproduce the results below use the following:

- Seed is set at: **2019**

- R Packages Used: **caret** , **randomForest** and **rpart**

**How the model was built** Our 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:

1. **Class A** - exactly according to the specification
2. **Class B** - throwing the elbows to the front
3. **Class C** - lifting the dumbbell only halfway
4. **Class D** - lowering the dumbbell only halfway
5. **Class E** - throwing the hips to the front

**Class A** corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes, see Human Activity Recognition Study.

**Model Selection:** Four models will be tested:

- **decision tree**
- **Generalized Boosted Regression** with 3-fold cross validation
- **random forest** with 3-fold and w/o cross validation

Prediction evaluation will be based on maximized accuracy and minimized out-of-sample error. The model with the highest accuracy will be chosen as our final model.

**Cross-validation**

There is a large sample size with N= 19,622 in the **Training** data set. This let us to divide our **Training** set for cross-validation using random subsamples of the **Training** data:

1. **Sub-Training** - 70% of the original **Training** data, randomly selected without replacement
2. **Sub-Testing** - 30% balance of original **Training** data

The models will be fitted on the **Sub-Training** data set, and tested on the **Sub-Testing** data for cross-validation.

Once the most accurate model is choosen, it will be tested on the original **Test** data set.

**Expected out-of-sample error**

The expected out-of-sample error will correspond to the quantity: 1-accuracy in the cross-validation data. Accuracy is the proportion of correct classified observation over the total sample in the **Sub-Testing** data set. Expected accuracy is the expected accuracy in the out-of-sample data set (i.e. **Testing** data set). Thus, the expected value of the out-of-sample error will be the expected number of missclassified observations/total observations in the **Testing** data set, which is the quantity: 1-accuracy found during the cross-validation.

## Purpose of Model Algorithm Choices

Our outcome variable **"classe"** is an unordered factor variable. Given our chosen modeling methods, after cleaning, all available variables will be used for prediction and our error type is chosen to be 1-accuracy.

Features with all missing values will be discarded as well as features that are irrelevant. All other features will be kept as relevant variables.

Decision tree and random forest algorithms are known for their ability of detecting the features that are important for classification. Generalized Boosted Regression (GBM) algorithms for classification also perform well, but typically less so than Random Forest and more so than standard Decision Trees. GBM will be run to see if it can perform as well or better and to confirm feature selection used in Random Forest. Feature selection is in these modeling methods are built into the process, so it is not so necessary to do extensive factor variable selection in model preparation.

## Project Code and Results

### Packages, Libraries, Seed

*R packages, and seed setting reproduceability, setting working directory*

```
set.seed(2019) # set seed for reproducibility

#Install Caret Package and library
# install.packages("caret")
library(caret)

## Warning: package 'caret' was built under R version 3.4.4

## Loading required package: lattice

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 3.4.4

#Install randomForest Package and library
# install.packages("randomForest")
library(randomForest) #Random forest for classification and regression

## Warning: package 'randomForest' was built under R version 3.4.4

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
#Install rpart Package and library
# install.packages("rpart")
library(rpart) # Regressive Partitioning and Regression trees
library(rpart.plot) # Decision Tree plot

## Warning: package 'rpart.plot' was built under R version 3.4.4

#load parallel to use all machine cores for maximum processing speed
library(doParallel)

## Warning: package 'doParallel' was built under R version 3.4.4

## Loading required package: foreach

## Warning: package 'foreach' was built under R version 3.4.4

## Loading required package: iterators

## Warning: package 'iterators' was built under R version 3.4.4

## Loading required package: parallel

# setting working directory
setwd("~/Documents/Personal/Coursera/Practical Machine Learning/Final
Project")
```

### Loading Data and Preparation

- First we want to retrieve the data sets and load into R.
- Then check the data for missing values and correct as necessary.
- Uneeded variables will be dropped

```r
# After downloading both training and test data sets
# Some missing values are coded as string "#DIV/0!" or "" or "NA" - these
will be changed to NA.
# We notice that both data sets contain columns with all missing values -
these will be deleted.

# Loading the training data set into my R session replacing all missing with
"NA"
trainingset <- read.csv("pml-training.csv", na.strings=c("NA","#DIV/0!", ""))
# Loading the testing data set
testingset <- read.csv("pml-testing.csv", na.strings=c("NA","#DIV/0!", ""))

# Check dimensions for number of variables and number of observations
dim(trainingset)

## [1] 19622   160

dim(testingset)

## [1]  20 160
```

```r
# Delete columns with all missing values
trainingset<-trainingset[,colSums(is.na(trainingset)) == 0]
testingset <-testingset[,colSums(is.na(testingset)) == 0]

# Some variables are irrelevant to our current project: user_name,
raw_timestamp_part_1, raw_timestamp_part_2, cvtd_timestamp, new_window, and
num_window (columns 1 to 7). We can delete these variables.
trainingset   <-trainingset[,-c(1:7)]
testingset    <- testingset[,-c(1:7)]

# and have a look at our new datasets:
dim(trainingset)

## [1] 19622    53

dim(testingset)

## [1] 20 53

head(trainingset,1)

##   roll_belt pitch_belt yaw_belt total_accel_belt gyros_belt_x gyros_belt_y
## 1      1.41       8.07    -94.4                3            0            0
##   gyros_belt_z accel_belt_x accel_belt_y accel_belt_z magnet_belt_x
## 1        -0.02          -21            4           22            -3
##   magnet_belt_y magnet_belt_z roll_arm pitch_arm yaw_arm total_accel_arm
## 1           599          -313     -128      22.5    -161              34
##   gyros_arm_x gyros_arm_y gyros_arm_z accel_arm_x accel_arm_y accel_arm_z
## 1           0           0       -0.02        -288         109        -123
##   magnet_arm_x magnet_arm_y magnet_arm_z roll_dumbbell pitch_dumbbell
## 1         -368          337          516      13.05217        -70.494
##   yaw_dumbbell total_accel_dumbbell gyros_dumbbell_x gyros_dumbbell_y
## 1    -84.87394                   37                0            -0.02
##   gyros_dumbbell_z accel_dumbbell_x accel_dumbbell_y accel_dumbbell_z
## 1                0             -234               47             -271
##   magnet_dumbbell_x magnet_dumbbell_y magnet_dumbbell_z roll_forearm
## 1              -559               293               -65         28.4
##   pitch_forearm yaw_forearm total_accel_forearm gyros_forearm_x
## 1         -63.9        -153                  36            0.03
##   gyros_forearm_y gyros_forearm_z accel_forearm_x accel_forearm_y
## 1               0           -0.02             192             203
##   accel_forearm_z magnet_forearm_x magnet_forearm_y magnet_forearm_z
## 1            -215              -17              654              476
##   classe
## 1      A

head(testingset,1)

##   roll_belt pitch_belt yaw_belt total_accel_belt gyros_belt_x gyros_belt_y
## 1       123         27    -4.75               20         -0.5        -0.02
##   gyros_belt_z accel_belt_x accel_belt_y accel_belt_z magnet_belt_x
```

```
## 1        -0.46          -38            69          -179            -13
##   magnet_belt_y magnet_belt_z roll_arm pitch_arm yaw_arm total_accel_arm
## 1           581          -382     40.7     -27.8     178              10
##   gyros_arm_x gyros_arm_y gyros_arm_z accel_arm_x accel_arm_y accel_arm_z
## 1       -1.65        0.48       -0.18          16          38          93
##   magnet_arm_x magnet_arm_y magnet_arm_z roll_dumbbell pitch_dumbbell
## 1         -326          385          481     -17.73748       24.96085
##   yaw_dumbbell total_accel_dumbbell gyros_dumbbell_x gyros_dumbbell_y
## 1      126.236                    9             0.64             0.06
##   gyros_dumbbell_z accel_dumbbell_x accel_dumbbell_y accel_dumbbell_z
## 1            -0.61               21              -15               81
##   magnet_dumbbell_x magnet_dumbbell_y magnet_dumbbell_z roll_forearm
## 1               523              -528               -56          141
##   pitch_forearm yaw_forearm total_accel_forearm gyros_forearm_x
## 1          49.3         156                  33            0.74
##   gyros_forearm_y gyros_forearm_z accel_forearm_x accel_forearm_y
## 1           -3.34           -0.59            -110             267
##   accel_forearm_z magnet_forearm_x magnet_forearm_y magnet_forearm_z
## 1            -149             -714              419              617
##   problem_id
## 1          1
```

### Partitioning the training data set to allow cross-validation

- The training data set contains 53 variables and 19622 obs after removing columns with missing
- The testing data set contains 53 variables and 20 obs.
- In order to perform cross-validation, the training data set is partionned into 2 sets: subTraining (70%) and subTest (30%). Random subsampling without replacement.

```
subsamples <- createDataPartition(y=trainingset$classe, p=0.7, list=FALSE)
subTraining <- trainingset[subsamples, ]
subTesting <- trainingset[-subsamples, ]
dim(subTraining)

## [1] 13737    53

dim(subTesting)

## [1] 5885   53

head(subTraining,1)

##   roll_belt pitch_belt yaw_belt total_accel_belt gyros_belt_x gyros_belt_y
## 1      1.41       8.07    -94.4                3            0            0
##   gyros_belt_z accel_belt_x accel_belt_y accel_belt_z magnet_belt_x
## 1        -0.02          -21            4           22            -3
##   magnet_belt_y magnet_belt_z roll_arm pitch_arm yaw_arm total_accel_arm
## 1           599          -313     -128      22.5    -161              34
##   gyros_arm_x gyros_arm_y gyros_arm_z accel_arm_x accel_arm_y accel_arm_z
## 1           0           0       -0.02        -288         109        -123
##   magnet_arm_x magnet_arm_y magnet_arm_z roll_dumbbell pitch_dumbbell
```

```
## 1         -368          337          516      13.05217        -70.494
##   yaw_dumbbell total_accel_dumbbell gyros_dumbbell_x gyros_dumbbell_y
## 1    -84.87394                   37                0            -0.02
##   gyros_dumbbell_z accel_dumbbell_x accel_dumbbell_y accel_dumbbell_z
## 1                0             -234               47             -271
##   magnet_dumbbell_x magnet_dumbbell_y magnet_dumbbell_z roll_forearm
## 1              -559               293               -65         28.4
##   pitch_forearm yaw_forearm total_accel_forearm gyros_forearm_x
## 1         -63.9        -153                  36            0.03
##   gyros_forearm_y gyros_forearm_z accel_forearm_x accel_forearm_y
## 1               0           -0.02             192             203
##   accel_forearm_z magnet_forearm_x magnet_forearm_y magnet_forearm_z
## 1            -215              -17              654              476
##   classe
## 1      A
```

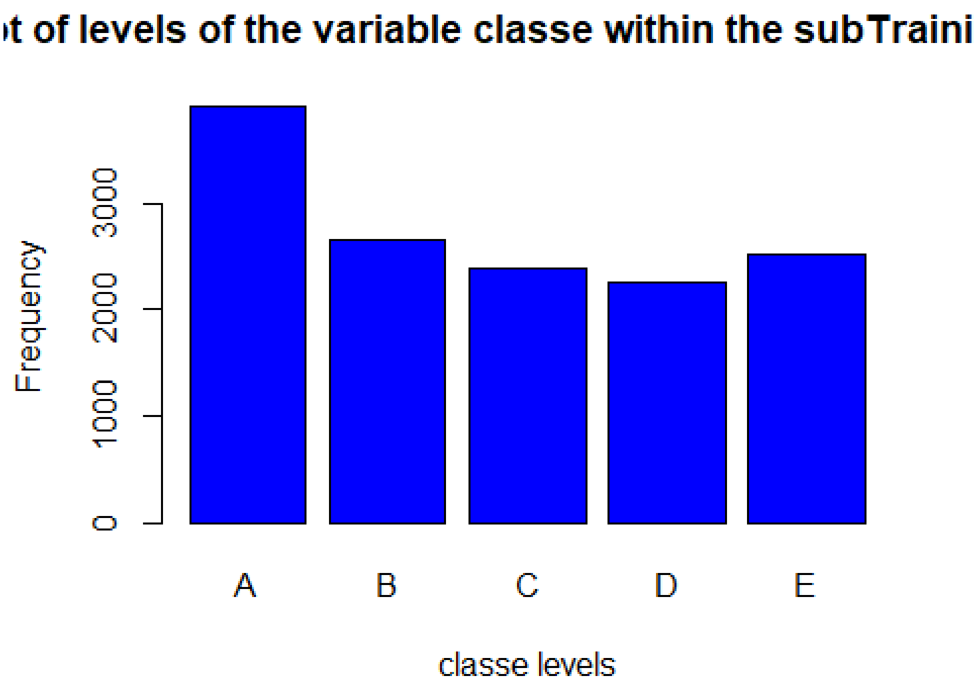
```
head(subTesting,1)
```

```
##   roll_belt pitch_belt yaw_belt total_accel_belt gyros_belt_x gyros_belt_y
## 5      1.48       8.07    -94.4                3         0.02         0.02
##   gyros_belt_z accel_belt_x accel_belt_y accel_belt_z magnet_belt_x
## 5        -0.02          -21            2           24            -6
##   magnet_belt_y magnet_belt_z roll_arm pitch_arm yaw_arm total_accel_arm
## 5           600          -302     -128      22.1    -161              34
##   gyros_arm_x gyros_arm_y gyros_arm_z accel_arm_x accel_arm_y accel_arm_z
## 5           0       -0.03           0        -289         111        -123
##   magnet_arm_x magnet_arm_y magnet_arm_z roll_dumbbell pitch_dumbbell
## 5         -374          337          506      13.37872      -70.42856
##   yaw_dumbbell total_accel_dumbbell gyros_dumbbell_x gyros_dumbbell_y
## 5    -84.85306                   37                0            -0.02
##   gyros_dumbbell_z accel_dumbbell_x accel_dumbbell_y accel_dumbbell_z
## 5                0             -233               48             -270
##   magnet_dumbbell_x magnet_dumbbell_y magnet_dumbbell_z roll_forearm
## 5              -554               292               -68           28
##   pitch_forearm yaw_forearm total_accel_forearm gyros_forearm_x
## 5         -63.9        -152                  36            0.02
##   gyros_forearm_y gyros_forearm_z accel_forearm_x accel_forearm_y
## 5               0           -0.02             189             206
##   accel_forearm_z magnet_forearm_x magnet_forearm_y magnet_forearm_z
## 5            -214              -17              655              473
##   classe
## 5      A
```

### A look at the Data

The variable “classe” contains 5 levels: A, B, C, D and E. A plot of the outcome variable will allow us to see the frequency of each levels in the subTraining data set and compare one another.

```r
plot(subTraining$classe, col="blue", main="Bar Plot of levels of the variable
classe within the subTraining data set", xlab="classe levels",
ylab="Frequency")
```




From the graph above, we can see that each level frequency is within the same order of magnitude of each other. Level A is the most frequent with more than 4000 occurrences while level D is the least frequent with about 2500 occurrences.

### Run Models on All Parallel Cores less 1

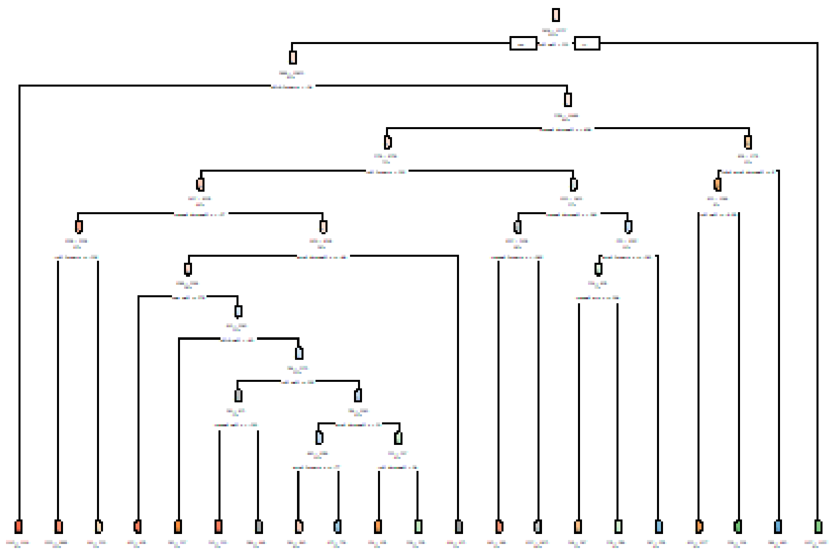
```r
library(doParallel)
cores <- detectCores() - 1
registerDoParallel(cores = cores)
```

## Class Prediction Model Evaluations

### First prediction model: Using Decision Tree

```r
# fitting Decision Tree model:
modeldt <- rpart(classe ~ ., data=subTraining, method="class")
# Predicting:
predict_dt <- predict(modeldt, subTesting, type = "class")

# Plot of the Decision Tree
rpart.plot(modeldt, main="Classification Tree", extra=102, under=TRUE,
faclen=0)
```

Classification Tree



```
# Test results on our subTesting data set:
confusionMatrix(predict_dt, subTesting$classe)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1507  185   70  124   35
##          B   55  671   89   83  102
##          C   51  137  778  144  124
##          D   32   84   72  522   52
##          E   29   62   17   91  769
##
## Overall Statistics
##
##                Accuracy : 0.7217
##                  95% CI : (0.71, 0.7331)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.646
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9002   0.5891   0.7583   0.5415   0.7107
```

```
## Specificity            0.9017   0.9307   0.9062   0.9512   0.9586
## Pos Pred Value         0.7845   0.6710   0.6305   0.6850   0.7944
## Neg Pred Value         0.9579   0.9042   0.9467   0.9137   0.9363
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2561   0.1140   0.1322   0.0887   0.1307
## Detection Prevalence   0.3264   0.1699   0.2097   0.1295   0.1645
## Balanced Accuracy      0.9010   0.7599   0.8322   0.7464   0.8346
```

### Second prediction model: Using GBM with Cross Validation (CV)

```
# guidance from https://topepo.github.io/caret/model-training-and-tuning.html

# The function trainControl can be used to specifiy the type of resampling:
fitControl <- trainControl(## 3-fold CV
                           method = "repeatedcv",
                           number = 3,
                           ## repeated ten times
                           repeats = 3)

# fitting GBM model using repeated cross-validation
set.seed(2019)
modelgbm <- train(classe ~. , data = subTraining,
                 method = "gbm",
                 metric = 'Accuracy',
                 trControl = fitControl,
                 ## This last option is actually one
                 ## for gbm() that passes through
                 verbose = FALSE)
print(modelgbm)
```

```
## Stochastic Gradient Boosting
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 3 times)
## Summary of sample sizes: 9158, 9159, 9157, 9158, 9158, 9158, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7530996  0.6868860
##   1                  100      0.8184705  0.7702565
##   1                  150      0.8536069  0.8147471
##   2                   50      0.8552568  0.8166181
##   2                  100      0.9050252  0.8797932
##   2                  150      0.9312561  0.9130163
##   3                   50      0.8953192  0.8674699
##   3                  100      0.9400402  0.9241359
##   3                  150      0.9598893  0.9492557
```

```
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
# Predicting GBM:
predict_gbm <- predict(modelgbm, subTesting)

# Test results on our subTesting data set:
confusionMatrix(predict_gbm, subTesting$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1650   26    0    0    1
##          B   19 1081   34    7   13
##          C    2   31  978   41    5
##          D    3    1   13  905   18
##          E    0    0    1   11 1045
##
## Overall Statistics
##
##                Accuracy : 0.9616
##                  95% CI : (0.9564, 0.9664)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9514
##  Mcnemar's Test P-Value : 3.88e-06
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9857   0.9491   0.9532   0.9388   0.9658
## Specificity            0.9936   0.9846   0.9837   0.9929   0.9975
## Pos Pred Value         0.9839   0.9367   0.9253   0.9628   0.9886
## Neg Pred Value         0.9943   0.9877   0.9901   0.9881   0.9923
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2804   0.1837   0.1662   0.1538   0.1776
## Detection Prevalence   0.2850   0.1961   0.1796   0.1597   0.1796
## Balanced Accuracy      0.9896   0.9668   0.9685   0.9658   0.9817
```

### Third prediction model: Using Random Forest with Cross Validation (CV)

```
# guidance from https://topepo.github.io/caret/model-training-and-tuning.html

# The function trainControl can be used to specifiy the type of resampling:
```

```r
fitControl <- trainControl(## 3 fold CV
                           method = "repeatedcv",
                           number = 3,
                           ## repeated ten times
                           repeats = 3)

# fitting Random Forest model using repeated cross-validation
set.seed(2019)
modelrf_cv <- train(classe ~. , data = subTraining,
                    method = "rf",
                    metric = 'Accuracy',
                    trControl = fitControl,
                    ## This last option is actually one
                    ## for gbm() that passes through
                    verbose = FALSE)
print(modelrf_cv)

## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 3 times)
## Summary of sample sizes: 9158, 9159, 9157, 9158, 9158, 9158, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9888622  0.9859091
##   27    0.9893475  0.9865242
##   52    0.9799810  0.9746731
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.

# Predicting Random Forest with CV:
predict_rf_cv <- predict(modelrf_cv, subTesting)

# Test results of Random Forest with CV on our subTesting data set:
confusionMatrix(predict_rf_cv, subTesting$classe)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    8    0    0    0
##          B    1 1129    4    0    1
##          C    0    2 1019   19    1
##          D    0    0    3  944    2
##          E    0    0    0    1 1078
```

```
##
## Overall Statistics
##
##                Accuracy : 0.9929
##                  95% CI : (0.9904, 0.9949)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.991
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9994   0.9912   0.9932   0.9793   0.9963
## Specificity            0.9981   0.9987   0.9955   0.9990   0.9998
## Pos Pred Value         0.9952   0.9947   0.9789   0.9947   0.9991
## Neg Pred Value         0.9998   0.9979   0.9986   0.9959   0.9992
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2843   0.1918   0.1732   0.1604   0.1832
## Detection Prevalence   0.2856   0.1929   0.1769   0.1613   0.1833
## Balanced Accuracy      0.9988   0.9950   0.9943   0.9891   0.9980
```

### Fourth prediction model: Using Random Forest without Cross Validation (CV)

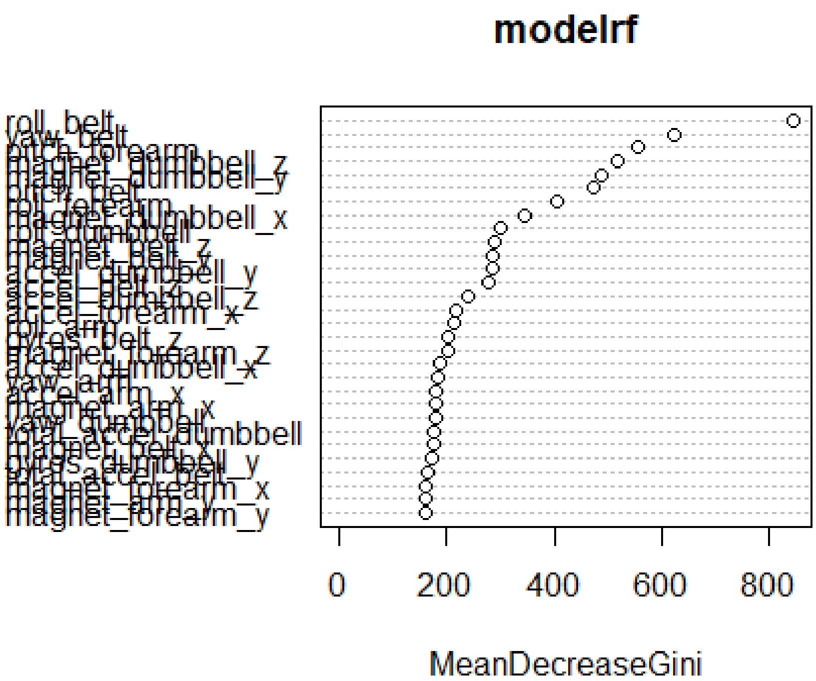
```
# fitting Random Forest model without cross-validation:
modelrf <- randomForest(classe ~. , data=subTraining, method="class")

# Print Random Forest without CV Model Summary:
print(modelrf)

##
## Call:
##  randomForest(formula = classe ~ ., data = subTraining, method = "class")
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.48%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3906    0    0    0    0 0.000000000
## B   12 2641    5    0    0 0.006395786
## C    0   17 2377    2    0 0.007929883
## D    0    0   20 2231    1 0.009325044
## E    0    0    2    7 2516 0.003564356

# Predicting Random Forest without CV:
predict_rf <- predict(modelrf, subTesting, type = "class")

# Test results of Random Forest without CV on our subTesting data set:
confusionMatrix(predict_rf, subTesting$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    4    0    0    0
##          B    0 1132    5    0    0
##          C    0    3 1021   18    0
##          D    0    0    0  945    3
##          E    0    0    0    1 1079
##
## Overall Statistics
##
##                Accuracy : 0.9942
##                  95% CI : (0.9919, 0.996)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9927
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9939   0.9951   0.9803   0.9972
## Specificity            0.9991   0.9989   0.9957   0.9994   0.9998
## Pos Pred Value         0.9976   0.9956   0.9798   0.9968   0.9991
## Neg Pred Value         1.0000   0.9985   0.9990   0.9962   0.9994
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1924   0.1735   0.1606   0.1833
## Detection Prevalence   0.2851   0.1932   0.1771   0.1611   0.1835
## Balanced Accuracy      0.9995   0.9964   0.9954   0.9898   0.9985
```

## Check Variable Importance Feature Selection in Random Forest and GBM Models

### Feature Importance for Random Forest Model (RF)

```
varImpPlot(modelrf,type=2)
```




```
imp <- as.data.frame(varImp(modelrf))
imp <- data.frame(names   = rownames(imp), overall = imp$Overall)
options(max.print=60)
imp[order(imp$overall,decreasing = T),]
```

```
##                  names   overall
## 1            roll_belt 846.79953
## 3             yaw_belt 622.05745
## 41       pitch_forearm 555.99780
## 39   magnet_dumbbell_z 519.37197
## 38   magnet_dumbbell_y 489.41431
## 2           pitch_belt 474.84953
## 40        roll_forearm 405.11769
## 37   magnet_dumbbell_x 345.37410
## 27       roll_dumbbell 298.86784
## 13       magnet_belt_z 287.95320
## 12       magnet_belt_y 284.18758
## 35    accel_dumbbell_y 283.04139
## 10        accel_belt_z 278.01073
## 36    accel_dumbbell_z 238.04721
## 47     accel_forearm_x 216.82963
## 14            roll_arm 212.95732
## 7          gyros_belt_z 202.73948
## 52    magnet_forearm_z 202.70708
## 34    accel_dumbbell_x 185.03034
## 16             yaw_arm 183.93375
## 21         accel_arm_x 180.74703
```

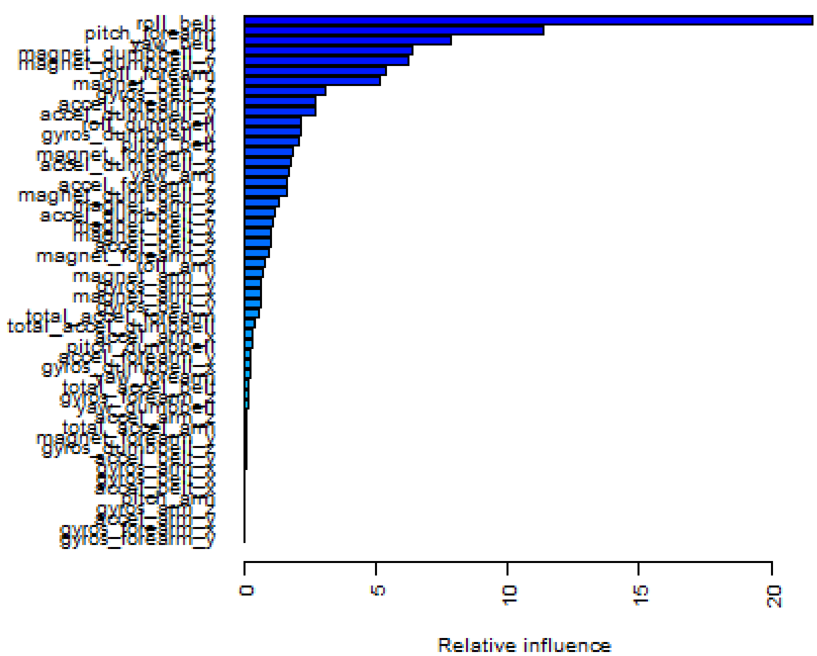
```
## 24         magnet_arm_x 179.37877
## 29         yaw_dumbbell 178.29186
## 30 total_accel_dumbbell 176.75253
## 11        magnet_belt_x 174.98958
## 32      gyros_dumbbell_y 171.44145
## 4      total_accel_belt 164.44861
## 50      magnet_forearm_x 160.67891
## 25         magnet_arm_y 160.30946
## 51      magnet_forearm_y 159.24562
##  [ reached getOption("max.print") -- omitted 22 rows ]
```

Top 5 Important features in predicting “classe” using Random Forest Model are found to be:

- roll_belt
- yaw_belt
- picth_forearm
- magnet_dumbbell_z
- magnet_dumbbell_y

### Feature Importance for Generalized Boosted Regression (GBM)

```r
par(las=2)
par(mar=c(5,15,4,2))
par(cex=0.60)
summary(modelgbm)
```



```
##                                       var     rel.inf
## roll_belt                       roll_belt 21.60134772
## pitch_forearm               pitch_forearm 11.35505760
## yaw_belt                         yaw_belt  7.84525005
## magnet_dumbbell_z       magnet_dumbbell_z  6.39595140
## magnet_dumbbell_y       magnet_dumbbell_y  6.22849845
## roll_forearm                 roll_forearm  5.37219762
## magnet_belt_z               magnet_belt_z  5.12447841
## gyros_belt_z                 gyros_belt_z  3.08520490
## accel_forearm_x           accel_forearm_x  2.64625477
## accel_dumbbell_y         accel_dumbbell_y  2.64197115
## roll_dumbbell               roll_dumbbell  2.16965558
## gyros_dumbbell_y         gyros_dumbbell_y  2.16598076
## pitch_belt                     pitch_belt  2.03308404
## magnet_forearm_z         magnet_forearm_z  1.85085760
## accel_dumbbell_x         accel_dumbbell_x  1.76489596
## yaw_arm                           yaw_arm  1.65684049
## accel_forearm_z           accel_forearm_z  1.61846945
## magnet_dumbbell_x       magnet_dumbbell_x  1.60991622
## magnet_arm_z                 magnet_arm_z  1.31447843
## accel_dumbbell_z         accel_dumbbell_z  1.17566339
## magnet_belt_y               magnet_belt_y  1.07607942
## magnet_belt_x               magnet_belt_x  1.00179020
## accel_belt_z                 accel_belt_z  0.97988748
## magnet_forearm_x         magnet_forearm_x  0.92626854
## roll_arm                         roll_arm  0.76126300
## magnet_arm_y                 magnet_arm_y  0.71405471
## gyros_arm_y                   gyros_arm_y  0.63330350
## magnet_arm_x                 magnet_arm_x  0.59698671
## gyros_belt_y                 gyros_belt_y  0.59680746
## total_accel_forearm   total_accel_forearm  0.52923165
##  [ reached getOption("max.print") -- omitted 22 rows ]
```

Top 5 Important features in predicting "classe" using GBM Model are found to be:

- roll_belt
- picth_forearm
- yaw_belt
- magnet_dumbbell_z
- magnet_dumbbell_y

In addition to both model algorithms performed well based on accuracy, feature importance ranking with Random Forest (RF) and Generalized Boosted Regression (GBM) models were very similar.

## Decision on Model Selection

As anticipated, the Random Forest algorithm performed better than a Decision Tree model and slightly better than Generalized Boosted Regression (GBM) model. Continued testing showed that the Random Forest model Accuracy was near 99.4% compared to near 72.2% for Decision Tree model and for GBM Model around 96.2%. The 4th model tested a Random Forest model (w/o 3-fold CV) is choosen due to simplicity and high accuracy using 500 Trees (ntree=500) and 7 Ranodm Variable Selection per iterration (mtry=7). The expected out-of-sample error is estimated at 0.006, or 0.6% calculated as 1 - accuracy for predictions made against the cross-validation set. The feature variable selection in our final model selection (Random Forest) is supported by feature importance findings in our Generalized Boosted Regression (GBM) model further validating our selected modle based on features for prediction.

Our evaluation **Test** data set comprises 20 cases and with an accuracy above 99% on our cross-validation data, we should see few, or none, of the test sample observations to be missclassified based on the Random Forest Model selected.

### Model Predition against Test Set Submission

```
# predict outcome levels on the original Testing data set using Random Forest
algorithm
predictfinal <- predict(modelrf, testingset, type="class")
predictfinal

##  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

# predictions to output files by observation for project submission
pml_write_files = function(x){
  n = length(x)
  path <- "answers"
  for(i in 1:n){
    filename = paste0("problem_id_",i,".txt")

write.table(x[i],file=file.path(path,filename),quote=FALSE,row.names=FALSE,co
l.names=FALSE)
  }
}

pml_write_files(predictfinal)
```