R Machine Learning Manual

Abu Nayeem

September 23, 2014

Introduction This Manual is meant to consolidate my knowledge in machine learning using the R package. The first two steps of machine learning is the following:

- What is the predictor variable [factor vs. numeric]
- How do you clean the dataset without removing valuable information?

Of course the standard practice of machine learning involves creating a training test [create model], cross validation set [test model], and a testing set [final unadulterated data]. This sort of testing procedure assures that the model does not face the issue of overfitting. This example will be a classification problem, but a numericla learning problem follow similar procedure.

Executive Summary Technology has focused on developing health tools and gadgets to record how much training a person has done in a specific period of time. However, almost no research has been done in developing tools or models to give the trainer feedback on how well he has been performing exercises. This project is oriented in calculating a machine learning algorithm to determine whether a weight lifting trainer performed the exercise well or made an error in the execution.

DataSet The data set used for the model comes from the Groupware@LES from their Human Activity Recognition project. They performed a study to analyze how well a Weight Lifting Exercise was executed. Each trainer was given a sensor for his glove, belt, dumbbell and arm-band. These are tools used by every weight lifting trainer so the original exercises maintain integrity.

Each trainer was asked to perform weight lifting in a particular manner. First, to do it perfectly as ideally described. Second, throwing the elbows to the front. Third, lifting the dumbbell half way. Fourth, lowering the dumbbell halfway. Finally, throwing the hips to the front. In each exercise performed, the sensors recorded the movements and rotations, including max accelerations, min accelerations, averages, kurtosis, between others.

You can learn more here http://groupware.les.inf.puc-rio.br/har

Preprocessing

```r
set.seed(234)
library(caret) # the power horse function; loads ggplot2 automatically
library(doMC) # enable parallel computing; loads parallel & iterators
library(nnet) # for neural networking and multi-nomial log regression models
library(randomForest) # random forest strategy
library(kernlab) # allows plenty of tools of dimension reduction and such
library(e1071) # allows more features but is needed for boosting models
library(plyr) # data table operations
library(dplyr) # data operations plus
library(gbm) # general boosting method
library(corrplot) # fancy correlation plot
```

```r
library(AppliedPredictiveModeling)
library(foreach) # used in random forest alogrithm
library(doParallel) # Parallel Processing
library(ipred) # needed for treebagging
library(rpart) # for rpart but it failed in this example
registerDoMC(cores = 2) # register the number of cores to parallel process
date() #set date
```

Loading libraries

```
## [1] "Fri Sep 26 00:13:16 2014"
```

```r
# Selecting the definition of NA string was defined via post-analysis
trainingfile <- 'http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv'
training <- read.csv(trainingfile, na.strings = c("NA", "#DIV/0!"))
training <- tbl_df(training) # this data table is smoother
testingfile <- 'http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv'
testing <- read.csv(testingfile, na.strings = c("NA", "#DIV/0!"))
testing <- tbl_df(testing)
```

Extraction

Setup An unresolved debate is do you keep the cross validation pristine prior to implementing data cleaning techiniques. I recommend testing your algorithm in both options to see if it actually makes a difference. The cross validation process lose precious data which can be used to create a prediction model. Let's give it the worse case scenario where cross-validation set is also treated as pristine. Note: you cannot fundamentally change the training set because the testing set is still raw , so you should take that to account.

Splitting training set into a smaller training set and cross-validation set

```r
inTrain <- createDataPartition(y = training$classe, p = 0.8, list = FALSE)
smalltraining <- training[inTrain, ]
crossvalidation <- training[-inTrain, ]
```

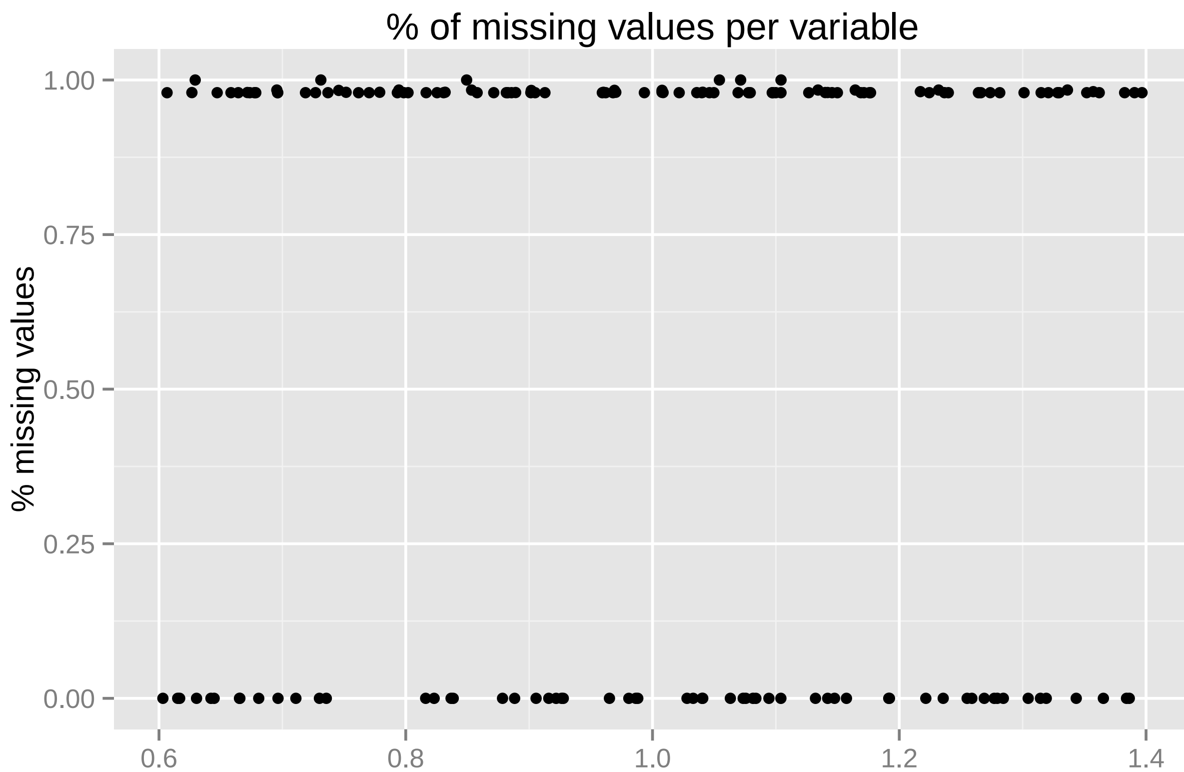
Data Cleaning

A) **Basics-** Make basic assessment that need to be done

```r
dim(training)
str(training)
summary(training)
```

B) **Handling missing values:** note you should set the strings for missing upon retriving the data
   Plotting of missing values:

```r
qplot(1, colSums(is.na(smalltraining))/dim(smalltraining)[1],
      geom = 'jitter',
      main = '% of missing values per variable',
      xlab = '', ylab = '% missing values') # visualization of missing values
```



Accumulative method Removal:

```
colSums(is.na(smalltraining)) # now we see the number of missing values in columns
                              # and see if they are significant for removal
NonNAIndex <- which(colSums(is.na(smalltraining)) > 0)
# this extracts the column index missing variable with number
RemoveNA <- smalltraining[ ,-NonNAIndex]
# Create new data frame that remmove columns that had missing values
```

Threshold Method: you choose the tolerable percentage and if above, remove the columns.

```
NonNAIndex <- which(apply(smalltraining, 2,
                          function(x) {sum(is.na(x))}) > 0.5 * dim(smalltraining)[1])
RemoveNA <- smalltraining[ ,-NonNAIndex]

# alternative to above but more function like
NA_threshold <- 0.50
nTrain <- nrow(smalltraining)
i <- 1
while(i < ncol(smalltraining)) {
  nNA <- sum(is.na(smalltraining[,i]))
  if((nNA/nTrain) >= NA_threshold) {
     NonNAIndex <- c(removeCols, i)
  }
  i <- i + 1
}
RemoveNA <- smalltraining[,-NonNAIndex]
```

C) **Removing Uninteresting Features:**

Removing Unrelated Features:

```
# choose the columns that may be useful for analysis
compacttraining <- select(RemoveNA, 2, 8:60)
```

Removing Near Zero variance features:

```
# this checks if all columns have close to zero variance
# the saveMetric provide heuristic information of each column which is REALLY useful
Nzv <- nearZeroVar(compacttraining,saveMetrics=TRUE)
Nzv # all false, so no columns will be removed
```

```
##                      freqRatio percentUnique zeroVar   nzv
## user_name                1.117       0.03822   FALSE FALSE
## roll_belt                1.103       7.54188   FALSE FALSE
## pitch_belt               1.032      11.14084   FALSE FALSE
## yaw_belt                 1.017      11.77782   FALSE FALSE
## total_accel_belt         1.052       0.17199   FALSE FALSE
## gyros_belt_x             1.038       0.83445   FALSE FALSE
## gyros_belt_y             1.119       0.42678   FALSE FALSE
## gyros_belt_z             1.062       1.04465   FALSE FALSE
## accel_belt_x             1.005       1.02554   FALSE FALSE
## accel_belt_y             1.105       0.87267   FALSE FALSE
## accel_belt_z             1.074       1.82814   FALSE FALSE
## magnet_belt_x            1.028       1.98102   FALSE FALSE
## magnet_belt_y            1.085       1.85362   FALSE FALSE
## magnet_belt_z            1.016       2.84095   FALSE FALSE
## roll_arm                55.204      16.04561   FALSE FALSE
## pitch_arm               84.562      18.50436   FALSE FALSE
## yaw_arm                 34.679      17.40238   FALSE FALSE
## total_accel_arm          1.050       0.42041   FALSE FALSE
## gyros_arm_x              1.012       4.05121   FALSE FALSE
## gyros_arm_y              1.415       2.36958   FALSE FALSE
## gyros_arm_z              1.108       1.51602   FALSE FALSE
## accel_arm_x              1.015       4.87292   FALSE FALSE
## accel_arm_y              1.131       3.38238   FALSE FALSE
## accel_arm_z              1.120       4.94936   FALSE FALSE
## magnet_arm_x             1.123       8.47825   FALSE FALSE
## magnet_arm_y             1.039       5.48443   FALSE FALSE
## magnet_arm_z             1.046       8.00688   FALSE FALSE
## roll_dumbbell            1.196      85.47678   FALSE FALSE
## pitch_dumbbell           2.515      83.41296   FALSE FALSE
## yaw_dumbbell             1.000      84.98630   FALSE FALSE
## total_accel_dumbbell     1.075       0.26116   FALSE FALSE
## gyros_dumbbell_x         1.023       1.51602   FALSE FALSE
## gyros_dumbbell_y         1.246       1.71985   FALSE FALSE
## gyros_dumbbell_z         1.061       1.27397   FALSE FALSE
## accel_dumbbell_x         1.046       2.61800   FALSE FALSE
## accel_dumbbell_y         1.145       2.89827   FALSE FALSE
## accel_dumbbell_z         1.189       2.57341   FALSE FALSE
## magnet_dumbbell_x        1.080       6.92401   FALSE FALSE
```

```
## magnet_dumbbell_y        1.157       5.27422   FALSE FALSE
## magnet_dumbbell_z        1.032       4.27416   FALSE FALSE
## roll_forearm            12.008      12.73330   FALSE FALSE
## pitch_forearm           63.120      17.47245   FALSE FALSE
## yaw_forearm             14.748      11.77782   FALSE FALSE
## total_accel_forearm      1.117       0.42678   FALSE FALSE
## gyros_forearm_x          1.017       1.80903   FALSE FALSE
## gyros_forearm_y          1.033       4.59902   FALSE FALSE
## gyros_forearm_z          1.101       1.87910   FALSE FALSE
## accel_forearm_x          1.013       5.00669   FALSE FALSE
## accel_forearm_y          1.132       6.25518   FALSE FALSE
## accel_forearm_z          1.008       3.60533   FALSE FALSE
## magnet_forearm_x         1.015       9.41461   FALSE FALSE
## magnet_forearm_y         1.203      11.79056   FALSE FALSE
## magnet_forearm_z         1.044      10.42105   FALSE FALSE
## classe                   1.469       0.03185   FALSE FALSE
```

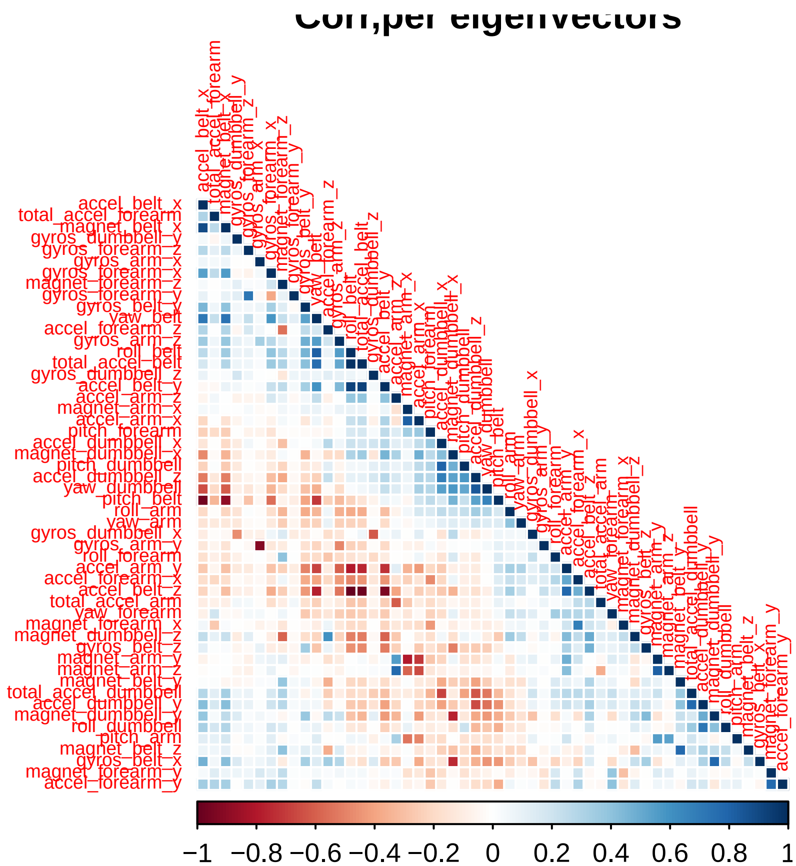
```
Nzv <- nearZeroVar(compacttraining,saveMetrics=FALSE)
```

```
# if there was columns to be removed this would be used
compacttraining <- compacttraining[ ,-Nzv]
```

D) **Removing Highly Correlated Variables:** For numerical/integer columns only

   Plotting Correlated Variables:

```
corData<- cor(compacttraining[ ,c(2:53)])
corrplot(corData,
         title = "Corr,per eigenVectors",
         order = "AOE",
         method = "color",
         type = "lower",
         tl.cex = 0.6 )  # plot to have a look to correlations
```



Removing Correlated Variables (Manual Method)

```
M <- abs(cor(compacttraining[ ,c(2:53)])) # create a correlation matrix
diag(M) <- 0 # by default the diagnols are one so we make them equal zero
which(M > 0.8, arr.ind = TRUE) # displays correlated pairs names
which(M > 0.8, arr.ind = FALSE) # displays the column numbers of each match
# you can remove certain pairs manually
descriptivetraining <- select(compacttraining,
                        -c(magnet_arm_y , pitch_dumbbell, yaw_dumbbell , accel_arm_x,
                           gyros_arm_y, pitch_belt, accel_belt_x, yaw_belt, total_accel_belt,
                           accel_belt_y, accel_belt_z, gyros_forearm_y,
                           gyros_dumbbell_z, gyros_dumbbell_x)) #40 variables left
```

Alternative Removal Method: Note this methos had42 variables left

```
descrCor <- cor(compacttraining[ ,c(2:53)])
highlyCorDescr <- findCorrelation(descrCor, cutoff = 0.8)
descriptivetraining <- compacttraining[, -highlyCorDescr] #42 varaibles left
```

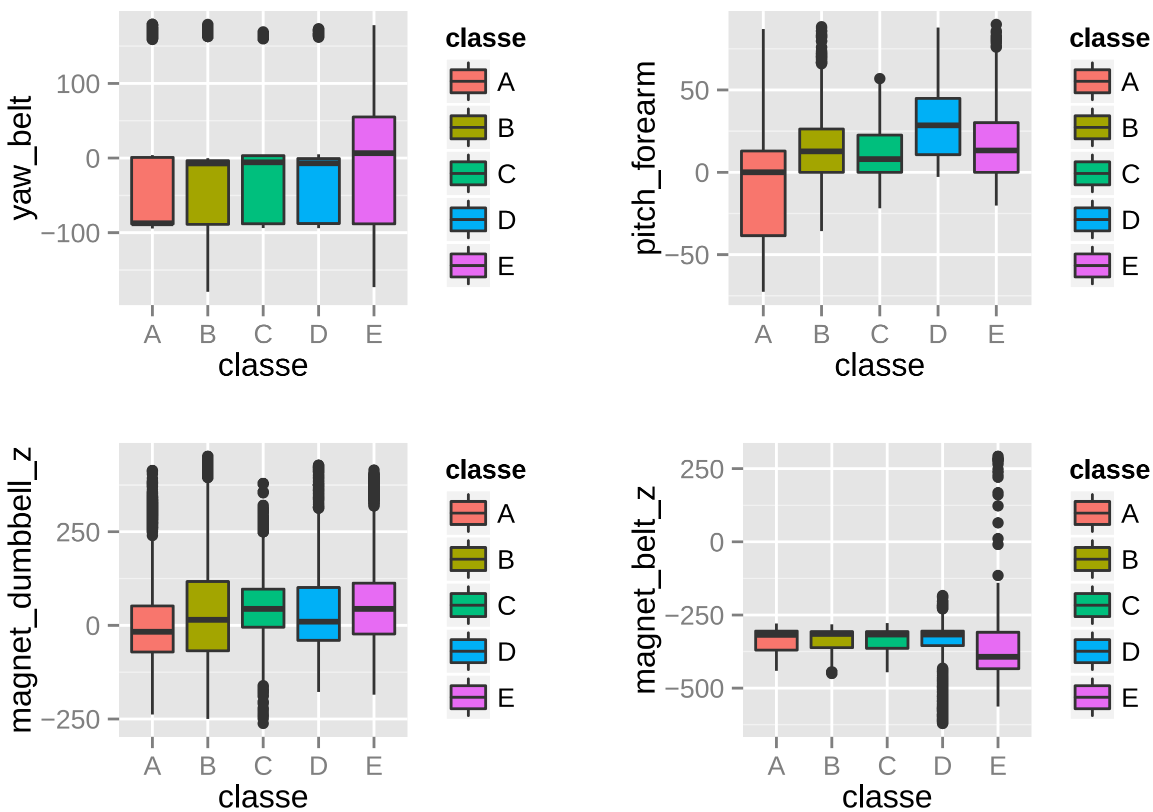
Removing high reasonable skewness [Not useful in predictions]: remember numerical columns only

```
factordescriptivetrain<-descriptivetraining[, c(1,40)] # separate non numerical variable
numdescriptivetrain<-descriptivetraining[, -c(1,40)] # separate numerical variables
NonskewIndex<-which(apply(numdescriptivetrain, 2,
```

```
                    function(x) abs(skewness(x)) > 6)) # find skewed volumns
numdescriptivetrain <- numdescriptivetrain[, -NonskewIndex] # remove skewed columns
cleandata <- cbind(factordescriptivetrain,numdescriptivetrain) # Combine to create clean data
```

E) **Exploratory Analysis:** [if you have an idea which variables are of concern]

```
require(gridExtra)
require(ggplot2)
p1 <- qplot(classe,yaw_belt,geom="boxplot",data=smalltraining,fill=classe)
p2 <- qplot(classe,pitch_forearm,geom="boxplot",data=smalltraining,fill=classe)
p3 <- qplot(classe,magnet_dumbbell_z,geom="boxplot",data=smalltraining,fill=classe)
p4 <- qplot(classe,magnet_belt_z,geom="boxplot",data=smalltraining,fill=classe)
grid.arrange(p1,p2,p3,p4, ncol=2)
```




```
colIndex <- colnames(cleandata) #38 variables should be remaining
check<-smalltraining[,colIndex]; check
# the colnames should be identical to that of cleandata
```

Complete the Column Index

Training Models **Pre-training:** Loading- You want to save your results so you don't need to constantly repeat analysis. Also note the rpart method failed.

```
if(file.exists("Machine Learning.RData")) {
  load("Machine Learning.RData")
}
```

Random Forests: Typically you may want build to smaller trees

The classe variable is actually a categorical variable and therefore a classification method performs better. One could use a single tree, but Random Forest have proven to be the most accurate classification algorithm, mainly for the reduction of variability while averaging different random trees. The out-of-bag (oob) error rate is important in this model:

In random forests, there is no need for cross-validation or a separate test set to get an unbiased estimate of the test set error. It is estimated internally, during the run, as follows: each tree is constructed using a different bootstrap sample from the original data. About one-third of the cases are left out of the bootstrap sample and not used in the construction of the kth tree. Put each case left out in the construction of the kth tree down the kth tree to get a classification. In this way, a test set classification is obtained for each case in about one-third of the trees. At the end of the run, take j to be the class that got most of the votes every time case n was oob. The proportion of times that j is not equal to the true class of n averaged over all cases is the oob error estimate. This has proven to be unbiased in many tests.

Method 1: Standard Random Forest model Run the model: [make sure parallel is running]

```
registerDoParallel()
Trfor1<- system.time(rf1 <- randomForest(classe ~ .,
                                         data = smalltraining[,colIndex],
                                         importance=TRUE))
```

```
Check Predictions:
```

```
rf1 # OOB estimate of  error rate: 0.77%
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = smalltraining[, colIndex],      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 0.77%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4460    4    0    0    0   0.0008961
## B   18 3009   11    0    0   0.0095458
## C    0   22 2710    6    0   0.0102264
## D    0    2   42 2527    2   0.0178780
## E    0    0    3   11 2872   0.0048510
```

```
rf1predictions1 <- predict(rf1, crossvalidation)
confusionMatrix(rf1predictions1,crossvalidation$classe)
```

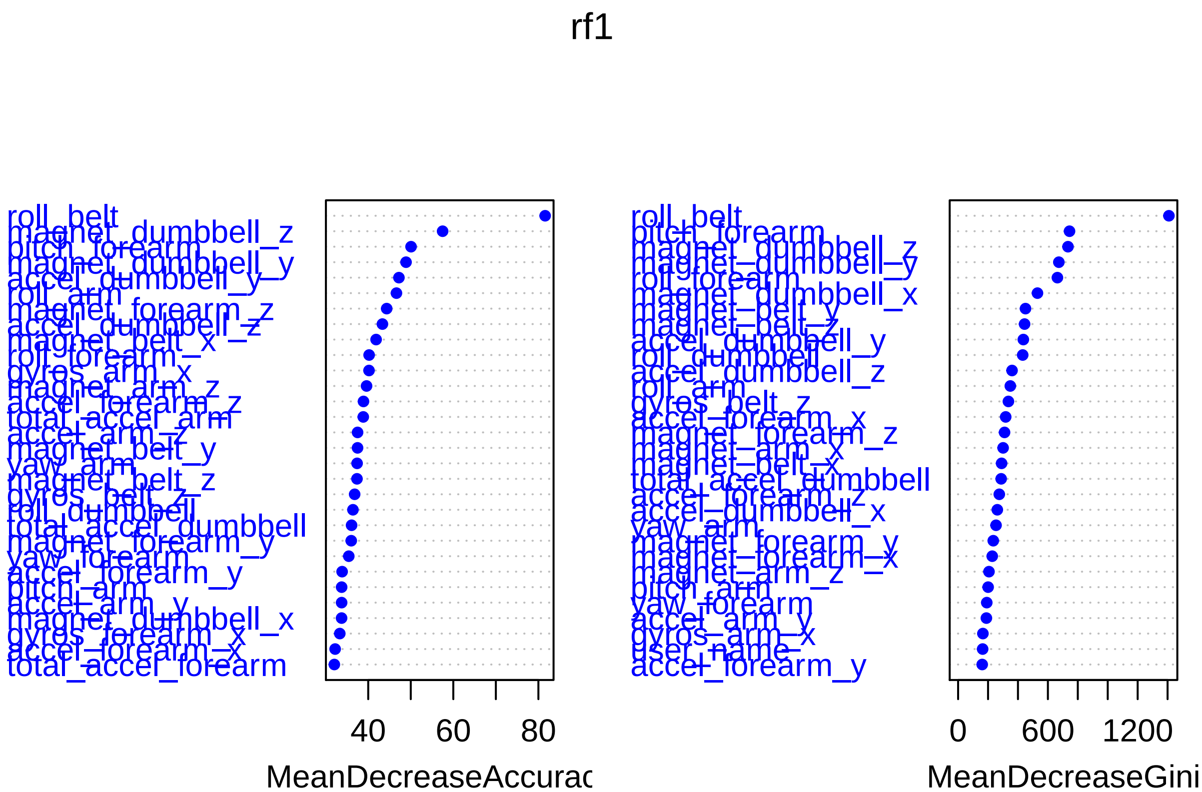
```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction    A    B    C    D    E
##          A 1114    2    0    0    0
##          B    2  754    6    0    0
##          C    0    3  678   11    0
##          D    0    0    0  632    2
##          E    0    0    0    0  719
##
## Overall Statistics
##
##                Accuracy : 0.993
##                  95% CI : (0.99, 0.996)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.992
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.998    0.993    0.991    0.983    0.997
## Specificity             0.999    0.997    0.996    0.999    1.000
## Pos Pred Value          0.998    0.990    0.980    0.997    1.000
## Neg Pred Value          0.999    0.998    0.998    0.997    0.999
## Prevalence              0.284    0.193    0.174    0.164    0.184
## Detection Rate          0.284    0.192    0.173    0.161    0.183
## Detection Prevalence    0.284    0.194    0.176    0.162    0.183
## Balanced Accuracy       0.999    0.995    0.993    0.991    0.999
```

```r
rfor1<- confusionMatrix(rf1predictions1,crossvalidation$classe)
```

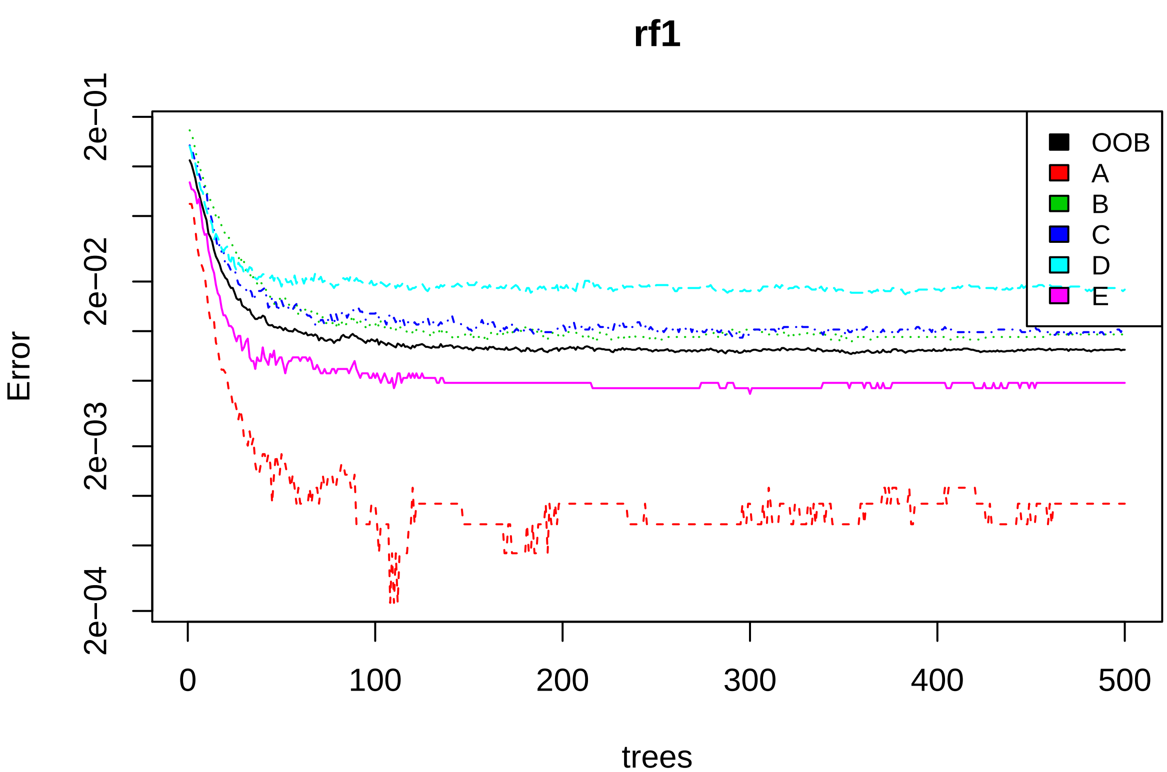
```
Assesment: What are the most influential trees? *Exclusive to random forest
```

```r
varImpPlot(rf1,pch=20,col="blue")
```



Plot: Choosing the right number of trees

```
plot(rf1, log="y")
legend("topright", colnames(rf1$err.rate),col=1:4,cex=0.8,fill=1:6)
```



Method 2: We now build 6 random forests with 150 trees each. We make use of parallel processing to build this model. Note: error with graphing tree

Set up and train model

```
t <- smalltraining[, colIndex]
x<- t[, -38]
y <- smalltraining$classe
Trfor2<- system.time(rf2 <- foreach(ntree=rep(150, 6),
                                    .combine=randomForest::combine,
                                    .packages='randomForest')
                     %dopar% {
                         randomForest(x, y, ntree=ntree)
                         })
```

Check Prediction

```
rf2 # OOB rate of 0% and used 900 trees
```

```
##
## Call:
##  randomForest(x = x, y = y, ntree = ntree)
##                Type of random forest: classification
##                      Number of trees: 900
## No. of variables tried at each split: 6
```

```
#we need to remove the missing values for this setup in
NonNAIndex <- which(colSums(is.na(crossvalidation)) > 0)
cross <- crossvalidation[ ,-NonNAIndex]

# cross is corssvalidation with missing variables missing
rf2predictions <- predict(rf2, cross)
confusionMatrix(rf2predictions,cross$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1116    0    0    0    0
##          B    0  759    0    0    0
##          C    0    0  684    0    0
##          D    0    0    0  643    0
##          E    0    0    0    0  721
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.999, 1)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 1
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             1.000    1.000    1.000    1.000    1.000
## Specificity             1.000    1.000    1.000    1.000    1.000
## Pos Pred Value          1.000    1.000    1.000    1.000    1.000
## Neg Pred Value          1.000    1.000    1.000    1.000    1.000
## Prevalence              0.284    0.193    0.174    0.164    0.184
## Detection Rate          0.284    0.193    0.174    0.164    0.184
## Detection Prevalence    0.284    0.193    0.174    0.164    0.184
## Balanced Accuracy       1.000    1.000    1.000    1.000    1.000
```

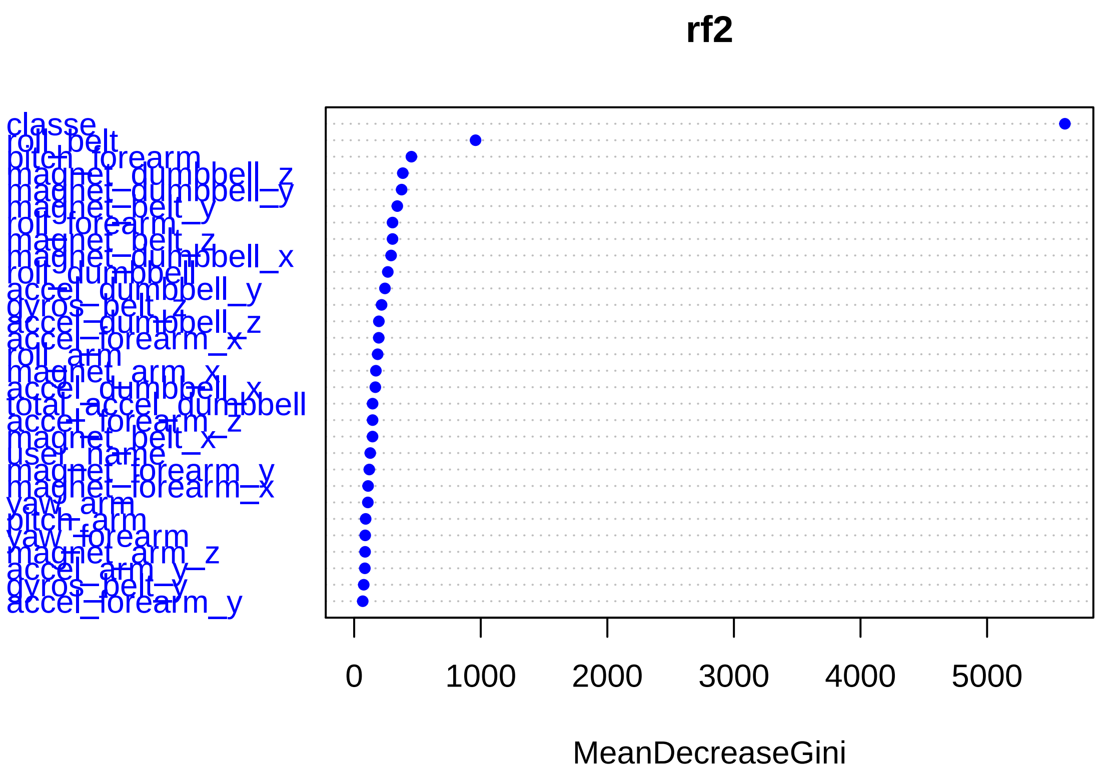
```r
rfor2<- confusionMatrix(rf2predictions,crossvalidation$classe)
# 100% accurate?

#testing respect to original test set
rf2predictions2 <- predict(rf2, RemoveNA)
confusionMatrix(rf2predictions2,RemoveNA$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 4464    0    0    0    0
##          B    0 3038    0    0    0
##          C    0    0 2738    0    0
##          D    0    0    0 2573    0
##          E    0    0    0    0 2886
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (1, 1)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             1.000    1.000    1.000    1.000    1.000
## Specificity             1.000    1.000    1.000    1.000    1.000
## Pos Pred Value          1.000    1.000    1.000    1.000    1.000
## Neg Pred Value          1.000    1.000    1.000    1.000    1.000
## Prevalence              0.284    0.194    0.174    0.164    0.184
## Detection Rate          0.284    0.194    0.174    0.164    0.184
## Detection Prevalence    0.284    0.194    0.174    0.164    0.184
## Balanced Accuracy       1.000    1.000    1.000    1.000    1.000
```

```r
#100% accurate?
```

```
Assesment: What are the most influential trees?
```

```r
varImpPlot(rf2,pch=20,col="blue")
```



SVM RADIAL: Support vector Machine is used for both classification and logistic regression. The radial kernal uses shortest distance of Euclidean distance. The kfolds separate the sample in two and then the model trains each section to predict the other; this information is then used to create the final model. Increased folds may increase validity but for each increased fold there is less data to predict the model. So be careful

- I customized `train` control function to perform k-fold cross validation of 2.

  Set up and run the model:

```r
tC <- trainControl(method = "cv", number = 2) # note 'cv' creates folds and 2 is the size
TSVMRad<- system.time(SVMRadial1 <- train(classe ~ .,
                                          method = "svmRadial",
                                          trControl = tC,
                                          data = smalltraining[, colIndex]))
```

```
Check predictions:
```

```r
SVMRadpredictions1 <- predict(SVMRadial1, crossvalidation)
confusionMatrix(SVMRadpredictions1,crossvalidation$classe)
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1107   72    1    3    5
##          B    5  651   38    4   17
##          C    4   28  638   73   36
##          D    0    2    2  563   19
##          E    0    6    5    0  644
##
## Overall Statistics
##
##                Accuracy : 0.918
##                  95% CI : (0.909, 0.927)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.897
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.992    0.858    0.933    0.876    0.893
## Specificity             0.971    0.980    0.956    0.993    0.997
## Pos Pred Value          0.932    0.910    0.819    0.961    0.983
## Neg Pred Value          0.997    0.966    0.985    0.976    0.976
## Prevalence              0.284    0.193    0.174    0.164    0.184
## Detection Rate          0.282    0.166    0.163    0.144    0.164
## Detection Prevalence    0.303    0.182    0.199    0.149    0.167
## Balanced Accuracy       0.982    0.919    0.945    0.934    0.945
```

```r
SVMRad <- confusionMatrix(SVMRadpredictions1,crossvalidation$classe)
```

SVM RADIAL COST: Similar to above but it now implements a penalty to reduce possibility of overfitting

`Setup and the run the model:`

```r
# model creation and test
tC <- trainControl(method = "cv", number = 2)
TSVMRadCost<- system.time(SVMRadialCost1 <- train(classe ~ .,
                                                  method = "svmRadialCost",
                                                  trControl = tC,
                                                  data = smalltraining[, colIndex]))
```

`Check predictions:`

```r
SVMRadCostpredictions1 <- predict(SVMRadialCost1, crossvalidation)
confusionMatrix(SVMRadCostpredictions1,crossvalidation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction    A    B    C    D    E
##          A 1106   72    1    3    5
##          B    5  650   38    4   17
##          C    4   28  638   73   36
##          D    1    2    2  563   19
##          E    0    7    5    0  644
##
## Overall Statistics
##
##                Accuracy : 0.918
##                  95% CI : (0.909, 0.926)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.896
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.991    0.856    0.933    0.876    0.893
## Specificity             0.971    0.980    0.956    0.993    0.996
## Pos Pred Value          0.932    0.910    0.819    0.959    0.982
## Neg Pred Value          0.996    0.966    0.985    0.976    0.976
## Prevalence              0.284    0.193    0.174    0.164    0.184
## Detection Rate          0.282    0.166    0.163    0.144    0.164
## Detection Prevalence    0.303    0.182    0.199    0.150    0.167
## Balanced Accuracy       0.981    0.918    0.945    0.934    0.945
```

```
SVMRadCost<- confusionMatrix(SVMRadCostpredictions1,crossvalidation$classe)
```

TREE BAG: it builds an expansive bundle of classification trees

Setup model and train it:

```
tC <- trainControl(method = "cv", number = 2)
TTB<- system.time(treebag1 <- train(classe ~ .,
                                    method = "treebag",
                                    trControl = tC,
                                    data = smalltraining[, colIndex]))
```

Check predictions:

```
treepredictions1 <- predict(treebag1, crossvalidation)
confusionMatrix(treepredictions1,crossvalidation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1110    9    0    2    0
##          B    4  736    9    1    1
```

```
##          C    0    9  670   18    2
##          D    0    5    3  620    4
##          E    2    0    2    2  714
##
## Overall Statistics
##
##                Accuracy : 0.981
##                  95% CI : (0.977, 0.985)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.976
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.995    0.970    0.980    0.964    0.990
## Specificity             0.996    0.995    0.991    0.996    0.998
## Pos Pred Value          0.990    0.980    0.959    0.981    0.992
## Neg Pred Value          0.998    0.993    0.996    0.993    0.998
## Prevalence              0.284    0.193    0.174    0.164    0.184
## Detection Rate          0.283    0.188    0.171    0.158    0.182
## Detection Prevalence    0.286    0.191    0.178    0.161    0.184
## Balanced Accuracy       0.995    0.982    0.985    0.980    0.994
```

```
TB<- confusionMatrix(treepredictions1,crossvalidation$classe)
```

Classification Tree: The most simplification form fo the classification tree

```
# apply classification tree
TCT<- system.time(Classtree1 <- train(classe ~ .,
                                      method="rpart",
                                      data = smalltraining[, colIndex]))
```

```
Classtreepredictions1 <- predict(Classtree1, crossvalidation)
confusionMatrix(Classtreepredictions1, crossvalidation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 999 321 327 280 106
##          B  28 247  26 108  91
##          C  84 191 331 255 201
##          D   0   0   0   0   0
##          E   5   0   0   0 323
##
## Overall Statistics
##
##                Accuracy : 0.484
##                  95% CI : (0.469, 0.5)
##     No Information Rate : 0.284
```

```
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.326
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.895    0.325   0.4839    0.000   0.4480
## Specificity             0.632    0.920   0.7743    1.000   0.9984
## Pos Pred Value          0.491    0.494   0.3117      NaN   0.9848
## Neg Pred Value          0.938    0.850   0.8766    0.836   0.8893
## Prevalence              0.284    0.193   0.1744    0.164   0.1838
## Detection Rate          0.255    0.063   0.0844    0.000   0.0823
## Detection Prevalence    0.518    0.127   0.2707    0.000   0.0836
## Balanced Accuracy       0.763    0.623   0.6291    0.500   0.7232
```

```
CT <- confusionMatrix(Classtreepredictions1, crossvalidation$classe)
```

Gradient Boosting (GBM)- is a machine learning technique for regression problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. The gradient boosting method can also be used for classification problems by reducing them to regression with a suitable loss function.

`Take a smaller sample to train model:`

```
sampletrain <- smalltraining[sample(nrow(smalltraining), 3000), ]
inTrain <- createDataPartition(y=sampletrain$classe, p=0.7, list=FALSE)
tinytraining <- sampletrain[inTrain, ]
tinycrossvalidation <- sampletrain[-inTrain, ]
```

`Set up the grid and run the model:`

```
gbmGrid <-  expand.grid(interaction.depth = 5, # the num of interactions between features
                        n.trees = 150, # the total number of trees or iterations
                        shrinkage = 0.1) # the learning rate of step-size function
TGBM<- system.time(GBM1 <- train(classe ~ .,
                                 method="gbm",
                                 data=tinytraining[ ,colIndex],
                                 tuneGrid = gbmGrid,
                                 verbose = FALSE)) #verbose doesn't show output iterations
```

`Check predictions:`

```
# test tiny crossvalidation
GBM1predictions<-predict(GBM1,tinycrossvalidation)
confusionMatrix(GBM1predictions,tinycrossvalidation$classe)
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   A   B   C   D   E
##          A 247  10   0   0   1
##          B   1 160   9   0   0
##          C   0   7 146   9   2
##          D   0   1   5 133   4
##          E   0   0   1   1 160
##
## Overall Statistics
##
##                Accuracy : 0.943
##                  95% CI : (0.926, 0.957)
##     No Information Rate : 0.276
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.928
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.996    0.899    0.907    0.930    0.958
## Specificity             0.983    0.986    0.976    0.987    0.997
## Pos Pred Value          0.957    0.941    0.890    0.930    0.988
## Neg Pred Value          0.998    0.975    0.980    0.987    0.990
## Prevalence              0.276    0.198    0.179    0.159    0.186
## Detection Rate          0.275    0.178    0.163    0.148    0.178
## Detection Prevalence    0.288    0.190    0.183    0.159    0.181
## Balanced Accuracy       0.990    0.942    0.941    0.958    0.978
```

```r
# test cross validation
GBM1predictions2<-predict(GBM1,crossvalidation)
confusionMatrix(GBM1predictions2,crossvalidation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1094   37    1    1    5
##          B   12  671   44    8   17
##          C    5   42  620   37   10
##          D    4    5   18  592   16
##          E    1    4    1    5  673
##
## Overall Statistics
##
##                Accuracy : 0.93
##                  95% CI : (0.922, 0.938)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.912
##  Mcnemar's Test P-Value : 5.33e-07
##
```

```
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.980    0.884    0.906    0.921    0.933
## Specificity             0.984    0.974    0.971    0.987    0.997
## Pos Pred Value          0.961    0.892    0.868    0.932    0.984
## Neg Pred Value          0.992    0.972    0.980    0.984    0.985
## Prevalence              0.284    0.193    0.174    0.164    0.184
## Detection Rate          0.279    0.171    0.158    0.151    0.172
## Detection Prevalence    0.290    0.192    0.182    0.162    0.174
## Balanced Accuracy       0.982    0.929    0.939    0.954    0.965
```

```
GBM<- confusionMatrix(GBM1predictions2,crossvalidation$classe)
```

Comparing Models

Measuring Accuracy and Out of Sammple Error:

```
# sum up all the methods
FinalAccuracy <- data.frame(rfor1$overall[1], rfor2$overall[1], SVMRad$overall[1],
                            SVMRadCost$overall[1], TB$overall[1], CT$overall[1],
                            GBM$overall[1])
colnames(FinalAccuracy) <- c("rfor1", "rfor2", "SVMRad", "SVMRadCost", "TB", "CT","GBM")
rownames(FinalAccuracy) <- "Accuracy"
FinalAccuracy
```

```
##           rfor1 rfor2 SVMRad SVMRadCost     TB     CT    GBM
## Accuracy 0.9934     1 0.9184     0.9179 0.9814 0.4843 0.9304
```

```
# show the out-of-sample error
outOfSamErr <- 1-FinalAccuracy
rownames(outOfSamErr) <- "OSError"
outOfSamErr
```

```
##            rfor1 rfor2  SVMRad SVMRadCost      TB     CT     GBM
## OSError 0.006628     0 0.08157    0.08208 0.01861 0.5157 0.06959
```

Measuring Kappa- Goodness to Fit

```
FinalKappa <- data.frame(rfor1$overall[2], rfor2$overall[2], SVMRad$overall[2],
                            SVMRadCost$overall[2], TB$overall[2], CT$overall[2],
                            GBM$overall[2])
colnames(FinalKappa) <- c("rfor1", "rfor2", "SVMRad", "SVMRadCost", "TB", "CT","GBM")
rownames(FinalKappa) <- "Kappa"
FinalKappa
```

```
##        rfor1 rfor2 SVMRad SVMRadCost     TB     CT    GBM
## Kappa 0.9916     1 0.8966     0.8959 0.9765 0.3262 0.9119
```

Measuring Size of each prediction model

```
FinalSize <- data.frame(format(object.size(rf1), units = "MB"),
                        format(object.size(rf2), units = "MB"),
                        format(object.size(SVMRadial1), units = "MB"),
                        format(object.size(SVMRadialCost1), units = "MB"),
                        format(object.size(treebag1), units = "MB"),
                        format(object.size(Classtree1), units = "MB"),
                        format(object.size(GBM1), units = "MB"))
colnames(FinalSize) <- c("rfor1", "rfor2", "SVMRad", "SVMRadCost", "TB", "CT","GBM")
rownames(FinalSize) <- "Size"
FinalSize
```

```
##         rfor1   rfor2 SVMRad SVMRadCost       TB      CT    GBM
## Size 31.3 Mb 54.3 Mb 9.4 Mb     9.4 Mb 187.8 Mb 12.6 Mb 3.5 Mb
```

Comparing computation time:

```
FinalComp <- rbind(Trfor1, Trfor2, TSVMRad, TSVMRadCost, TTB, TCT, TGBM)
rownames(FinalComp) <- c("rfor1", "rfor2", "SVMRad", "SVMRadCost", "TB", "CT","GBM")
FinalComp
```

```
##            user.self sys.self elapsed user.child sys.child
## rfor1         95.100    1.620  113.29       0.00     0.000
## rfor2          1.737    1.110  158.65      66.99    11.475
## SVMRad        32.607    5.600  521.33      70.22     2.798
## SVMRadCost    30.673    4.399  468.57     195.30    33.380
## TB            52.930    1.151  138.17      95.85    13.183
## CT             7.076    1.056  365.35      54.72     4.868
## GBM            5.978    0.220   89.47      94.46     3.838
```

Complete Model Comparison:

```
Group <- rbind(FinalKappa, outOfSamErr, FinalSize)
TGroup<- data.frame(t(Group)) # transform to matrix and transpose it
CompleteComparison<- data.frame(cbind(TGroup,FinalComp))
CompleteComparison <- mutate(CompleteComparison, usertime=user.self + user.child,
                             systime=sys.self + sys.child,
                             model = c("rfor1", "rfor2", "SVMRad", "SVMRadCost",
                                       "TB", "CT","GBM"))
CompleteComparison<- CompleteComparison[, -c(4,5,7,8)]
CompleteComparison[, 1] <- round(as.numeric(as.character(CompleteComparison[, 1])), 3)
CompleteComparison[, 2] <- round(as.numeric(as.character(CompleteComparison[, 2])), 3)
CompleteComparison <- select(CompleteComparison,7,1:6)
CompleteComparison <- arrange(CompleteComparison, OSError)
CompleteComparison # without timestamp variables
```

```
##        model Kappa OSError     Size elapsed usertime systime
## 1      rfor2 1.000   0.000  54.3 Mb  158.65    68.73  12.585
## 2      rfor1 0.992   0.007  31.3 Mb  113.29    95.10   1.620
## 3         TB 0.976   0.019 187.8 Mb  138.17   148.78  14.334
## 4        GBM 0.912   0.070   3.5 Mb   89.47   100.44   4.058
## 5     SVMRad 0.897   0.082   9.4 Mb  521.33   102.82   8.398
## 6 SVMRadCost 0.896   0.082   9.4 Mb  468.57   225.97  37.779
## 7         CT 0.326   0.516  12.6 Mb  365.35    61.80   5.924
```

Sub-comparison of excluding timestamp variables Sub-Comparison to when we include timestamp variables [it was significant]

```
first
```

```
##        models Kappa OSError     Size elapsed usertime
## 1       rfor2 1.000   0.000  46.3 Mb  156.82    77.85
## 2       rfor1 0.998   0.001  26.9 Mb  146.02    94.88
## 3          TB 0.989   0.009 189.5 Mb   86.87   148.61
## 4         GBM 0.979   0.016   3.6 Mb   90.56   111.69
## 5      SVMRad 0.904   0.075   9.8 Mb  970.34   173.54
## 6 SVMRadCost 0.903   0.076   9.9 Mb  286.08   125.25
## 7          CT 0.326   0.516    13 Mb   48.24   111.21
```

Notice including timestamp variables decrease total size for almost all algorithms. Some models are impacted significantly computationally when including them while others enjoy one less variable. The reason is that it provides more possibilites to match and separate the data OR it make it easy to reach the goal because of fewer variables. REGARDLESS it is worth to explore tradeoffs in more well-tweaked models.

Conclusion I've tested many machine learning models in this exercise. Normally, we just choose the most accurate algorithm and move on, but we need to consider the entire pipeline of the project. Several factors that we should care about is accuracy/outof sample error, fitted train model size, elapsed and system time. With that said the top three models are randomforest models, general boosting models, and treebag model. The treebag requires so much data to hold, so let's discard that. The GBM and randomforest are very both good candidates. Randomforest models have many additional features that shed a lot more of the internal processes, which can allow to build a more efficent model (less trees or remove the least interesting features. In contrast, GBM excels greatly in minimum size and training time while still maintaining accuaracy.

With the comparison chart feature you can do short diagnostic on which model you want to implement. Note the logistic regression woould use a similar procedure with a few differences.

Bibliography 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.

Additional Material **Principal Component Analysis-** It can only handle numeric vectors

```
preProcompact <- preProcess(compacttraining[,-c(1,57)], method="pca")
preProcdescriptive <- preProcess(descriptivetraining[,-c(1,42)], method="pca")
```

Do NOT Use PCA to modelfit for large datasets as it crashed for R or taken enormous amount of time to complete.

Per person approach

Train classifier for training subset Based on the findings from the previous section, we'll learn separate predictor for each user.

Apply model to test subset and determine out of sample error The out-of sample error seems to be well under control.

Train classifier on full test set