Regression Models: Peer Assessment

Introduction

Machine Learning: Peer Assessment
Author: Tim Westran
Purpose: Determine how well exercises are performed Data set used: The mtcars data set.

Practical Machine Learning - Prediction Assignment Writeup

Synopsis

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).

Load libraries and set options

First, we want to load libraries and set options.

```r
## Ensure we have loaded the libraries we need.
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```r
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```r
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

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

```r
library(rpart)
library(rpart.plot)
library(rattle)
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
##
## Attaching package: 'rattle'

## The following object is masked from 'package:randomForest':
##
##     importance
```

```
library(doParallel)
```

```
## Loading required package: foreach

## Loading required package: iterators

## Loading required package: parallel
```

```
## disable scientific notation
options(scipen = 999)
options(warn=-1)
registerDoParallel(cores=4)
```

Download files and prepare data

```
## Download the training & test data and read it as csv file:
if (!file.exists("pmlTraining.csv")) {
    download.file("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",
        destfile = "pmlTraining.csv")
}
if (!file.exists("pmlTesting.csv")) {
    download.file("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",
        destfile = "pmlTesting.csv")
}

## Read in the training and test data sets.
training <- read.csv("pmlTraining.csv", header = TRUE, na.strings = c("NA",""))
testing <- read.csv("pmlTesting.csv", header = TRUE, na.strings = c("NA",""))

## Create our training and test sets.  70% of samples to training, 30% to test.
labels <- createDataPartition(training$classe, p = 0.7, list = FALSE)
train <- training[labels, ]
test <- training[-labels, ]
```

Data Preprocessing

Some of the variables have next to no variance, and others have many NA data points. We can remove variables with low variance, variables with many NAs, and any variables which don't provide much information about the data.

```
## Grab all variables from the training set that have low variability.
NearZero <- nearZeroVar(train)

## Remove these low variability variables from both the training and test sets.
train <- train[ ,-NearZero]
test <- test[ ,-NearZero]

## Find all variables which have many NA values
labels <- apply(train, 2, function(x) mean(is.na(x))) > 0.95
```

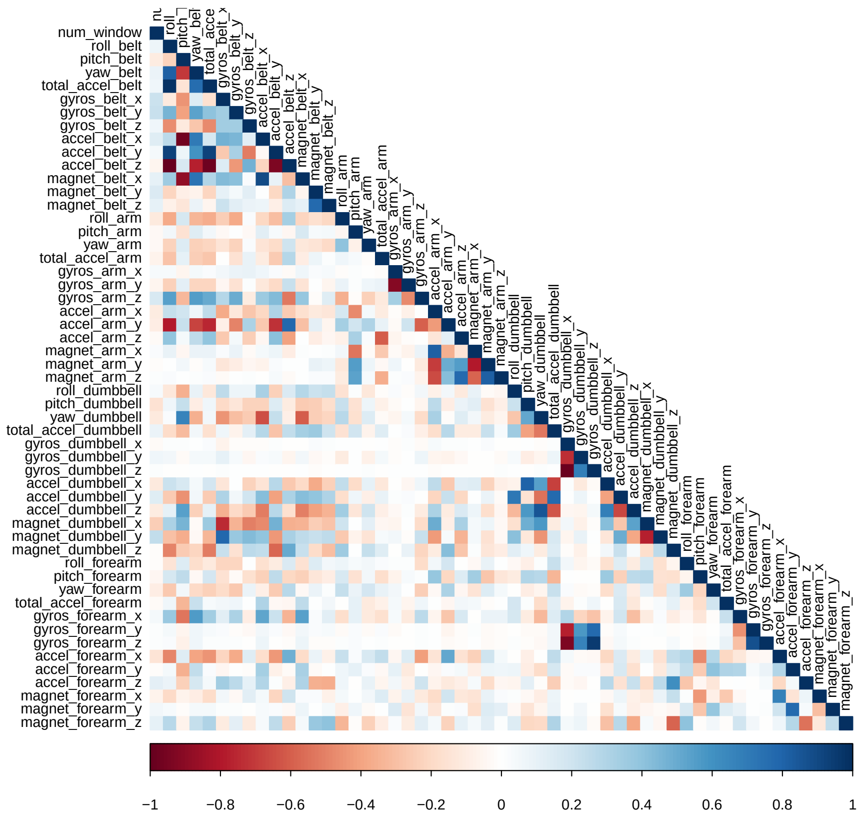
```
## Remove the high NA variables from our training and test sets.
train <- train[, -which(labels, labels == FALSE)]
test <- test[, -which(labels, labels == FALSE)]

## Remove unneeded columns.
train <- train[ , -(1:5)]
test <- test[ , -(1:5)]
```

By removing the low variance variables, variables with many NAs, and purely descriptive columns, we reduce the total number of variables from 160 to 54.

Exploratory Analysis

First, we use a correlation plot to investigate variable dependence. For this plot, darker colors mean higher correlation.

```
corrplot(cor(train[,-54]), method = "color", type = "lower", tl.cex = 0.8, tl.col = rgb(0,0,0))
```



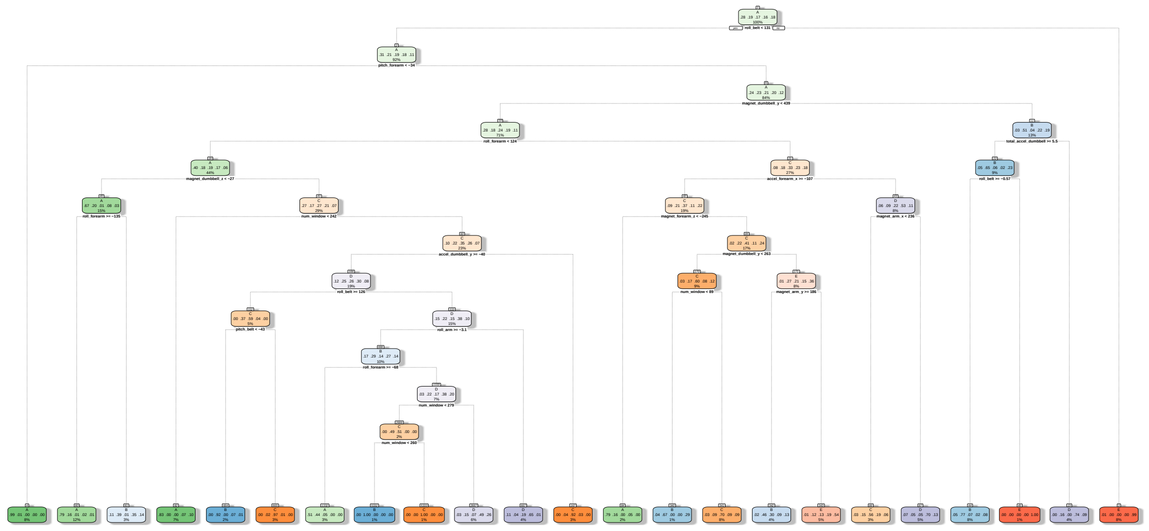
Prediction Model Selection

We will pick from 3 different methods to predict variables. We will look at decision trees, random forests, and generalized boosted models.

Decision Tree

```
## First, set the seed.  Picking a random number.
set.seed(444)
```

```r
## Next, create the decision tree.
DecisionTreeModel <- rpart(classe ~ ., data = train, method = "class")

##  Plot the decision tree.
fancyRpartPlot(DecisionTreeModel)
```



Rattle 2019−Mar−06 12:52:42 Laptop1

```r
## Last, show a confusion matrix.
DecisionTreePredictor <- predict(DecisionTreeModel, test, type = "class")
DecisionTreeConfusionMatrix <- confusionMatrix(DecisionTreePredictor, test$classe)
DecisionTreeConfusionMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1538  219   16   42   48
##          B   53  694  103   78  122
##          C   14   70  783   81   53
##          D   58  107   87  703  160
##          E   11   49   37   60  699
##
## Overall Statistics
##
##                Accuracy : 0.7506
##                  95% CI : (0.7393, 0.7616)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.6834
##  Mcnemar's Test P-Value : < 0.00000000000000022
##
```

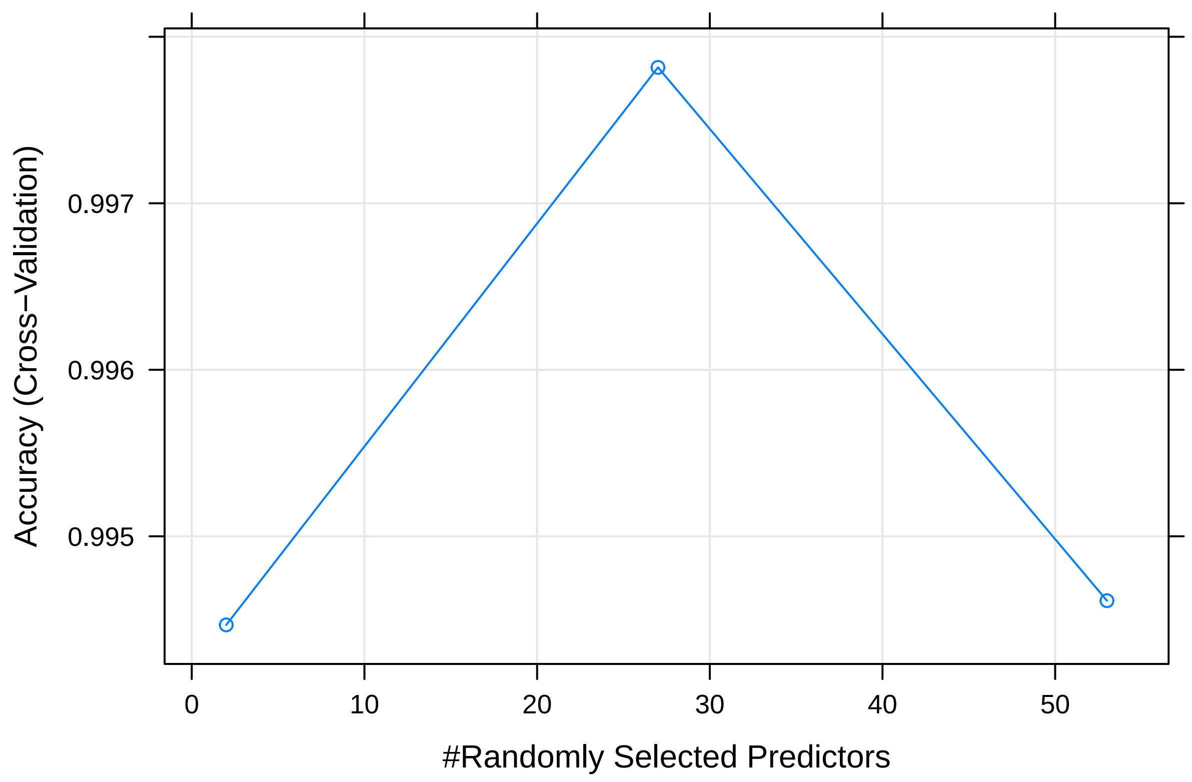
```
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9188   0.6093   0.7632   0.7293   0.6460
## Specificity            0.9228   0.9250   0.9551   0.9163   0.9673
## Pos Pred Value         0.8256   0.6610   0.7822   0.6305   0.8166
## Neg Pred Value         0.9662   0.9080   0.9502   0.9453   0.9238
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2613   0.1179   0.1331   0.1195   0.1188
## Detection Prevalence   0.3166   0.1784   0.1701   0.1895   0.1455
## Balanced Accuracy      0.9208   0.7671   0.8591   0.8228   0.8067
```

Random Forest

```
## First, set the seed.  Picking a random number.
set.seed(444)

## Use 10-Fold Cross Validation to calculate errors.
control <- trainControl(method = "cv", number = 10, verboseIter=FALSE)
model <- train(classe ~ ., data = train, method = 'rf', trControl = control)
model
```

```
## Random Forest
##
## 13737 samples
##    53 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 12364, 12363, 12363, 12363, 12363, 12363, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9944677  0.9930022
##   27    0.9978161  0.9972377
##   53    0.9946132  0.9931857
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.
```

```
plot(model)
```



Using cross validation, we estimate the Out of sample error rate to be: 0.276% (using 27 variables)

```
## Next, create the random forest.
RandomForestModel <- train(classe ~ ., data = train, method = "rf", trControl = control)
RandomForestModel$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 0.2%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3905    0    0    0    1 0.0002560164
## B    4 2651    2    1    0 0.0026335591
## C    0    4 2392    0    0 0.0016694491
## D    0    0    6 2245    1 0.0031083481
## E    0    1    0    7 2517 0.0031683168
```

```
## Last, show a confusion matrix.
RandomForestPredictor <- predict(RandomForestModel, test)
RandomForestConfusionMatrix <- confusionMatrix(RandomForestPredictor, test$classe)
RandomForestConfusionMatrix
```

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

```
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    3    0    0    0
##          B    1 1136    3    0    0
##          C    0    0 1020    2    0
##          D    0    0    3  962    3
##          E    0    0    0    0 1079
##
## Overall Statistics
##
##                Accuracy : 0.9975
##                  95% CI : (0.9958, 0.9986)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.9968
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9994   0.9974   0.9942   0.9979   0.9972
## Specificity            0.9993   0.9992   0.9996   0.9988   1.0000
## Pos Pred Value         0.9982   0.9965   0.9980   0.9938   1.0000
## Neg Pred Value         0.9998   0.9994   0.9988   0.9996   0.9994
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2843   0.1930   0.1733   0.1635   0.1833
## Detection Prevalence   0.2848   0.1937   0.1737   0.1645   0.1833
## Balanced Accuracy      0.9993   0.9983   0.9969   0.9984   0.9986
```

Generalized Boosted Model

```
## First, set the seed.  Picking a random number.
set.seed(444)

## Next, create the generalized boosted model.
control <- trainControl(method = "repeatedcv", number = 5, repeats = 1, verboseIter = FALSE)
GeneralizedBoostModel <- train(classe ~ ., data = train, trControl = control, method = "gbm", verbose =
GeneralizedBoostModel$finalModel
```

```
## A gradient boosted model with multinomial loss function.
## 150 iterations were performed.
## There were 53 predictors of which 53 had non-zero influence.
```

```
## Last, show a confusion matrix.
GeneralizedBoostPredictor <- predict(GeneralizedBoostModel, test)
GeneralizedBoostConfusionMatrix <- confusionMatrix(GeneralizedBoostPredictor, test$classe)
GeneralizedBoostConfusionMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1666    8    0    0    0
##          B    8 1122    7    4    5
##          C    0    9 1018    5    5
##          D    0    0    1  954    5
##          E    0    0    0    1 1067
##
## Overall Statistics
##
##                Accuracy : 0.9901
##                  95% CI : (0.9873, 0.9925)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.9875
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9952   0.9851   0.9922   0.9896   0.9861
## Specificity            0.9981   0.9949   0.9961   0.9988   0.9998
## Pos Pred Value         0.9952   0.9791   0.9817   0.9937   0.9991
## Neg Pred Value         0.9981   0.9964   0.9983   0.9980   0.9969
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2831   0.1907   0.1730   0.1621   0.1813
## Detection Prevalence   0.2845   0.1947   0.1762   0.1631   0.1815
## Balanced Accuracy      0.9967   0.9900   0.9941   0.9942   0.9930
```

The Random Forest offers the highest accuracy, so we will use the random forest model for our prediction algoritm.

Predicting Test Set Output

```
RandomForestPredictor <- predict(RandomForestModel, testing)
RandomForestPredictor
```

```
##  [1] B A B A A E D B A A B C B A E E A B B B
## Levels: A B C D E
```