Practical Machine Learning

April 2015

Summary

Our class was tasked with looking at a data set containing raw data around performing certain personal activities in a specific way. The data included six subjects that performed an activity five different ways. The project goal is to predict the manner in which the exercise was performed. After cleaning the data and removing unnecessary columns, I used random forests and was able to successfully predict all 20 observations in our testing data set.

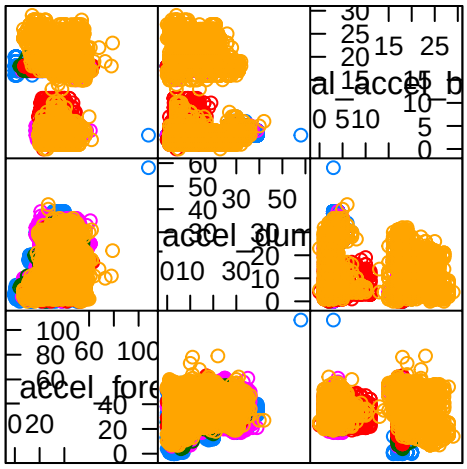
Data Loading & Exploration

First things first, I had to load the data.

```
url <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
url2 <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
download.file(url, dest = "training.csv")
download.file(url2, dest = "testing.csv")
trainingP <- read.csv("training.csv")
testingP <- read.csv("testing.csv")
```

After loading, I explored the data set to know what I had and what needed cleaning. I did a summary on the data, and will not display that here because it was large, and saw the variables and what type they were and what values they had. Once that was done, I started doing a little exploring and plotting different graphs. One graph showed the 3 totals columns against the classe, as seen below. The different colors represent the different classes. As you can see they all seem to be around each other so other variables must be in included in the prediction in order to accurately choose the correct classe.

```
require(caret)
featurePlot(x=trainingCleaned[,c("total_accel_forearm", "total_accel_dumbbell", "total_accel_belt")],
            y = trainingCleaned$classe, plot = "pairs")
```



Scatter Plot Matrix

Cleaning & Partitioning Data

Before building a model, I need to take the raw data and clean it and partition it so that I can predict.

cleaning the Data

First I removed all columns that had a NA in it since train doesn't work with NAs in the dataset. Then I looked to see if there were columns with less than 5% (or 1000 values) in the data. I removed those columns because they would not be a good prediction on the outcomes of classe with so few values present. Finally, I removed the first 7 columns as that related to some metadata. By doing this processing, I went from 160 variables to 53.

```
trainingCleaned <- trainingP[,colSums(is.na(trainingP)) == 0]
trainingCleaned <- trainingCleaned[, colSums(trainingCleaned != "") > 1000]
trainingCleaned <- trainingCleaned[, -(1:7)]
```

Partitioning the Data

Even though the data came with a training and test set, I do not want to touch my test set until I run the final model against it. Therefore, I am going to do a little cross validation. For me to run different models and find the best fit, I need to separate out my training data set into a sub-training and sub-test set so I can build a model on my sub-training set and apply to my sub-test set (a sample from the training data set) to see how my model works or if I need to tweak more. I split this training data set into 75% and 25% respectively.

```
require(caret)
trainIndex <- createDataPartition(trainingCleaned$classe, p=.75, list = FALSE)
trainingP1 <- trainingCleaned[trainIndex, ]
testingP1 <- trainingCleaned[-trainIndex,]
```

Building the Model

After looking at regression models like a decision tree and random forests, I chose to finish this assignment using the random forests as it provided the best solution. Using the decision trees, I ended up with 49% accuracy with my sub-test predictions, whereas using random forests I got a 99% accuracy. Also, knowing that random forests are known for their accuracy models, it made sense to apply that model for this assignment. Since I built my model using the randomForest function, I didn't have to do cross validation like I did above because cross validation is inherent in the function processing.

```
require(randomForest)
set.seed(318)
modelRF <- randomForest(classe ~ ., data = trainingP1)
predictions <- predict(modelRF, newdata=testingP1)
confusionMatrix(predictions, testingP1$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1394    4    0    0    0
##          B    0  944    4    0    0
##          C    0    1  851   10    2
##          D    0    0    0  794    4
##          E    1    0    0    0  895
##
## Overall Statistics
##
##                Accuracy : 0.9947
##                  95% CI : (0.9922, 0.9965)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9933
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9993   0.9947   0.9953   0.9876   0.9933
## Specificity            0.9989   0.9990   0.9968   0.9990   0.9998
## Pos Pred Value         0.9971   0.9958   0.9850   0.9950   0.9989
## Neg Pred Value         0.9997   0.9987   0.9990   0.9976   0.9985
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2843   0.1925   0.1735   0.1619   0.1825
## Detection Prevalence   0.2851   0.1933   0.1762   0.1627   0.1827
## Balanced Accuracy      0.9991   0.9969   0.9961   0.9933   0.9965
```

As stated before, with this model, I get an accuracy of 99%. In addition to the error rate I talk about a little later, I could also look at the kappa value that is a common error measure. Kappa is a measurement that compares observed accuracy, those observations that were classified correctly using the function, with Expected Accuracy, the number of observations in the class. The higher kappa value, the smaller the error rate in the model. In my case, with a very high kappa, my error rate is relatively small.

```
##
```

```
## Call:
##  randomForest(formula = classe ~ ., data = trainingP1)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.49%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4185    0    0    0    0 0.000000000
## B   13 2829    6    0    0 0.006671348
## C    0   17 2549    1    0 0.007012076
## D    0    0   26 2384    2 0.011608624
## E    0    0    1    6 2699 0.002586844
```

Also, I get an out-of-bag error rate of 0.49%. This error rate is a classification error based on an unbiased estimate of the error as more and more trees are added to the forest.

Conclusion

I was able to successfully clean the data and build a random forest model that was highly accurate and produced correct results for the testing samples that we had to turn into submission.