Human Activity Analysis

Glenn Tisdale

Thursday, December 21, 2014

Executive Summary

In this discussion I present the development of a random forest model for predicting Human Activity based on acceleromter data. The final model has an accuracy better than 97%.

The algorithm was developed using by creating intial test algorithms on a 75%-25% training-test data split. Then three different algorithms - simple tree, random forest and boosted trees - were tried to determine which algorithm was most applicable and which variables were most significant.

Then a model was created which used the top 20 variables and the top 10 variables against a test set that consisted of 10% of the data. This was used to select a compact feature set.

Finally the 20 variable feature set was applied to a random set of 20% of the original data to create a predictive model. This choice was made to make the running time for the model tractable.

The final predicted values are:

A A B A A E D B A A B C B A E E A B A B

Set up the Environment

The first step that I take is install any packages that are required, clear the evironment of old objects and then load the libraries from the required packages. Note that the package install lines are commented out since the packages have been loaded but they have been retainted for reference.

```
#install.packages("ada")
#install.packages("gbm")
#install.packages("caret")
#install.packages("plyr")
#install.packages("randomForest")
#update.packages(checkBuilt=TRUE)
rm(list=ls())
setwd("C:/Google Drive/71) Education/43) Coursera/Data Science - Johns Hopkins University/08 Practica
l Machine Learning/Project")
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.1.2
```

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

```
## Warning: package 'ggplot2' was built under R version 3.1.2
```

```r
library(ada)
```

```
## Warning: package 'ada' was built under R version 3.1.2
```

```
## Loading required package: rpart
```

```r
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.1.2
```

```
## Loading required package: survival
## Loading required package: splines
##
## Attaching package: 'survival'
##
## The following object is masked from 'package:caret':
##
##     cluster
##
## Loading required package: parallel
## Loaded gbm 2.1
```

```r
library(plyr)
```

```
## Warning: package 'plyr' was built under R version 3.1.2
```

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.1.2
```

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

Load the Data

The next step that I take is to load the raw training data files from the working directory and then set then rename the index columns with a simple index name.

```
training.raw <- read.csv("pml-training.csv")
names(training.raw)[1] <- "index"
testing.raw <- read.csv("pml-testing.csv")
names(testing.raw)[1] <- "index"
```

Clean the data

These particular data sets have numerous columns which that do not contain viable data. In many cases these columns are primarily empty or contain NA elements. The most direct way to do this is to examine the data with a spreadsheet (in this case Excel) and then select only those colunmns which contain data that is useful for the analysis. After loading the data, I run a check to insure that all NA elements have been elminated. The lines for this check have been commented out but are retained for reference.

```
cols <- c(160, 8:11, 37:49, 60:68, 84:86, 113:124, 151:159)
training.select <- training.raw[,cols]
testing.select <- testing.raw[,cols]
#which(is.na(training))
#which(is.na(testing))
```

Create Inital Models

The first analysis step that I take is to run a set of different algorithms on the data set to determine which type of algorithm will be the most useful.

Create Test and Training Sets

To enable the algorithms to be tested and cross validated, I divide the inital data into training and testing data - training received 75% and testing receives 25%.

```
set.seed(100)
inTrain <- createDataPartition(y=training.select$classe, p=.75, list=FALSE)
training <- training.select[inTrain,]
testing <- training.select[-inTrain,]
```

Create Mini Test and Training Sets

For this particular data set, there is so much data that running algorithms such as random forests will take a great deal of running time. Therefore I create abbreviated versions of the data sets which have a random selection of 10 percent of the training data and 10 percent of the test data.

```
training.idx <- which(!is.na(training$classe))
testing.idx <- which(!is.na(testing$classe))
in.mini.training <- sample(training.idx, size = ceiling(length(training.idx)/10))
in.mini.testing <- sample(testing.idx, size = ceiling(length(testing.idx)/10))
mini.training <- training[in.mini.training,]
mini.testing <- testing[in.mini.testing,]
```

Run Candidate Algorithms

To understand the performance of a variety of algorithms I run a simple tree model, a random forests model and a boosted trees model. For each model I generate a confusion matrix and a variable importance list.

What can quickly be seen from the models below is that a simple tree model generates better than random performance, but it looks like it could be improved upon. The random forest model by contrast generates a higher quality model with accuracy approximately 92%. The boosted tree model is almost as good as the random forest model with an accuracy of 89%

Note that for the random forest model and the boosted tree model, I have cached models from previous runs and have reloaded these models to eliminate the running time for each. The commented lines have been retained for reference.

Simple Tree

```
modFit <- train(classe ~ .,method = "rpart", data=mini.training)
print(modFit$finalModel)
```

```
## n= 1472
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 1472 1051 A (0.29 0.2 0.18 0.17 0.17)
##     2) roll_belt< 129.5 1348  933 A (0.31 0.22 0.19 0.18 0.098)
##       4) pitch_forearm< -34.15 123    1 A (0.99 0.0081 0 0 0) *
##       5) pitch_forearm>=-34.15 1225  931 B (0.24 0.24 0.21 0.2 0.11)
##        10) yaw_belt>=168.5 71    7 A (0.9 0.085 0 0.014 0) *
##        11) yaw_belt< 168.5 1154  866 B (0.2 0.25 0.23 0.21 0.11)
##          22) magnet_dumbbell_z< -39.5 323  194 A (0.4 0.34 0.056 0.18 0.028)
##            44) magnet_dumbbell_x< -434.5 132   32 A (0.76 0.2 0 0 0.038) *
##            45) magnet_dumbbell_x>=-434.5 191  108 B (0.15 0.43 0.094 0.3 0.021) *
##          23) magnet_dumbbell_z>=-39.5 831  587 C (0.12 0.21 0.29 0.22 0.15)
##            46) pitch_belt< -42.9 59    6 B (0 0.9 0.085 0.017 0) *
##            47) pitch_belt>=-42.9 772  533 C (0.13 0.16 0.31 0.24 0.16)
##              94) accel_dumbbell_y< -39.5 67    6 C (0.015 0.015 0.91 0.06 0) *
##              95) accel_dumbbell_y>=-39.5 705  524 D (0.14 0.18 0.25 0.26 0.17)
##               190) magnet_dumbbell_y< 300.5 323  177 C (0.15 0.11 0.45 0.16 0.13) *
##               191) magnet_dumbbell_y>=300.5 382  252 D (0.13 0.24 0.084 0.34 0.21) *
##     3) roll_belt>=129.5 124    6 E (0.048 0 0 0 0.95) *
```

```
pred <- predict(modFit, newdata=mini.testing)
confusionMatrix(pred,mini.testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  A  B  C  D  E
##          A 89 12  0  1  0
##          B 10 37  9 26  1
##          C 23 16 67 22 21
##          D 11 26  9 40 33
##          E  1  0  0  0 37
##
## Overall Statistics
##
##                Accuracy : 0.55
##                  95% CI : (0.505, 0.595)
##     No Information Rate : 0.273
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.438
##  Mcnemar's Test P-Value : 5.62e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.664   0.4066    0.788   0.4494   0.4022
## Specificity             0.964   0.8850    0.798   0.8035   0.9975
## Pos Pred Value          0.873   0.4458    0.450   0.3361   0.9737
## Neg Pred Value          0.884   0.8676    0.947   0.8683   0.8786
## Prevalence              0.273   0.1853    0.173   0.1813   0.1874
## Detection Rate          0.181   0.0754    0.136   0.0815   0.0754
## Detection Prevalence    0.208   0.1690    0.303   0.2424   0.0774
## Balanced Accuracy       0.814   0.6458    0.793   0.6265   0.6998
```

```
varImp(modFit)
```

```
## rpart variable importance
##
##   only 20 most important variables shown (out of 50)
##
##                   Overall
## roll_belt          100.00
## pitch_forearm       66.52
## magnet_dumbbell_x   66.48
## gyros_dumbbell_y    48.09
## roll_dumbbell       34.94
## magnet_dumbbell_y   34.50
## pitch_belt          28.95
## magnet_dumbbell_z   28.89
## accel_dumbbell_y    27.72
## accel_arm_x         27.58
## accel_belt_z        25.32
## magnet_belt_y       22.69
## total_accel_belt    20.95
## roll_forearm        16.97
## yaw_belt            16.96
## magnet_arm_x        16.74
## accel_forearm_x     11.03
## accel_dumbbell_x     8.27
## pitch_dumbbell       8.20
## gyros_belt_x         0.00
```

Random Forest

```
#modFit.all.rf.mini <- train(classe ~ .,method = "rf", data=mini.training, prox=TRUE)
#save(modFit.all.rf.mini,file="modFit_all_rf_mini")
load("modFit_all_rf_mini")
pred <- predict(modFit.all.rf.mini, newdata=mini.testing)
confusionMatrix(pred,mini.testing$classe)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   A   B   C   D   E
##          A 132   3   1   1   0
##          B   0  79   7   2   0
##          C   0   8  74   7   1
##          D   2   1   3  78   2
##          E   0   0   0   1  89
## 
## Overall Statistics
##                                         
##                Accuracy : 0.921         
##                  95% CI : (0.893, 0.943)
##     No Information Rate : 0.273         
##     P-Value [Acc > NIR] : <2e-16        
##                                         
##                   Kappa : 0.9           
##  Mcnemar's Test P-Value : NA            
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.985    0.868    0.871    0.876    0.967
## Specificity             0.986    0.978    0.961    0.980    0.997
## Pos Pred Value          0.964    0.898    0.822    0.907    0.989
## Neg Pred Value          0.994    0.970    0.973    0.973    0.993
## Prevalence              0.273    0.185    0.173    0.181    0.187
## Detection Rate          0.269    0.161    0.151    0.159    0.181
## Detection Prevalence    0.279    0.179    0.183    0.175    0.183
## Balanced Accuracy       0.986    0.923    0.916    0.928    0.982
```

```
varImp(modFit.all.rf.mini)
```

```
## rf variable importance
##
##                    Overall
## roll_belt          100.00
## yaw_belt            66.69
## magnet_dumbbell_z   63.59
## magnet_dumbbell_y   57.90
## pitch_forearm       53.92
## magnet_dumbbell_x   46.36
## pitch_belt          43.98
## roll_forearm        33.58
## accel_dumbbell_y    21.71
## accel_dumbbell_z    21.53
## roll_dumbbell       19.76
## magnet_belt_z       19.24
## accel_belt_z        18.83
## magnet_belt_y       16.11
## accel_forearm_x     13.93
## roll_arm            11.30
## gyros_dumbbell_y    11.22
## magnet_forearm_z    10.14
## gyros_belt_z         2.95
## magnet_forearm_x     0.00
```

Boosted Tree

```
#modFit.all.gbm.mini <- train(classe ~ .,method = "gbm", data=mini.training, verbose=FALSE)
#save(modFit.all.gbm.mini,file="modFit_all_gbm_mini")
load("modFit_all_gbm_mini")
pred <- predict(modFit.all.gbm.mini, newdata=mini.testing)
confusionMatrix(pred,mini.testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 127   6   4   0   0
##          B   1  75   7   2   0
##          C   2  10  72   4   5
##          D   4   0   2  80   2
##          E   0   0   0   3  85
##
## Overall Statistics
##
##                Accuracy : 0.894
##                  95% CI : (0.863, 0.92)
##     No Information Rate : 0.273
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.866
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.948    0.824    0.847    0.899    0.924
## Specificity             0.972    0.975    0.948    0.980    0.992
## Pos Pred Value          0.927    0.882    0.774    0.909    0.966
## Neg Pred Value          0.980    0.961    0.967    0.978    0.983
## Prevalence              0.273    0.185    0.173    0.181    0.187
## Detection Rate          0.259    0.153    0.147    0.163    0.173
## Detection Prevalence    0.279    0.173    0.189    0.179    0.179
## Balanced Accuracy       0.960    0.900    0.898    0.939    0.958
```

```
varImp(modFit.all.gbm.mini)
```

```
## gbm variable importance
## 
##                    Overall
## roll_belt         100.000
## pitch_forearm      62.379
## yaw_belt           55.758
## magnet_dumbbell_z  39.354
## magnet_dumbbell_y  32.069
## roll_forearm       25.559
## magnet_belt_z      20.968
## accel_dumbbell_y   19.421
## gyros_dumbbell_y   18.672
## pitch_belt         18.245
## magnet_dumbbell_x  17.438
## magnet_forearm_z   15.665
## roll_dumbbell      12.455
## gyros_belt_z       11.107
## accel_forearm_x     9.508
## accel_dumbbell_z    6.854
## magnet_forearm_x    4.721
## roll_arm            4.127
## magnet_belt_y       0.601
## accel_belt_z        0.000
```

Make predictions from the test data.

Finally I make a set of predictions from the random forest model agains the full set of testing data.

```
pred <- predict(modFit.all.rf.mini, newdata=testing.select)
pred
```

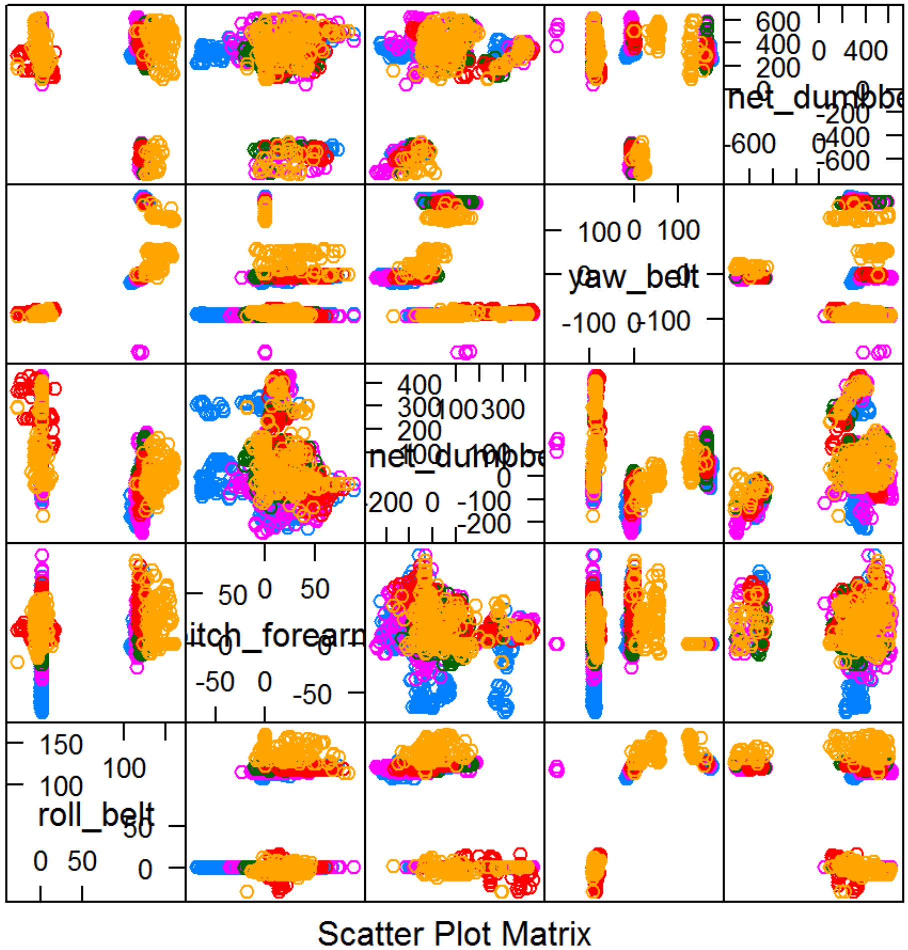
```
##  [1] C A B A A E D D A A B C B A E E A B B B
## Levels: A B C D E
```

This completes the inital algorithm evaluation. Now the goal will be to improve the random forest algorithm, which the above suggests as the most useful candidate.

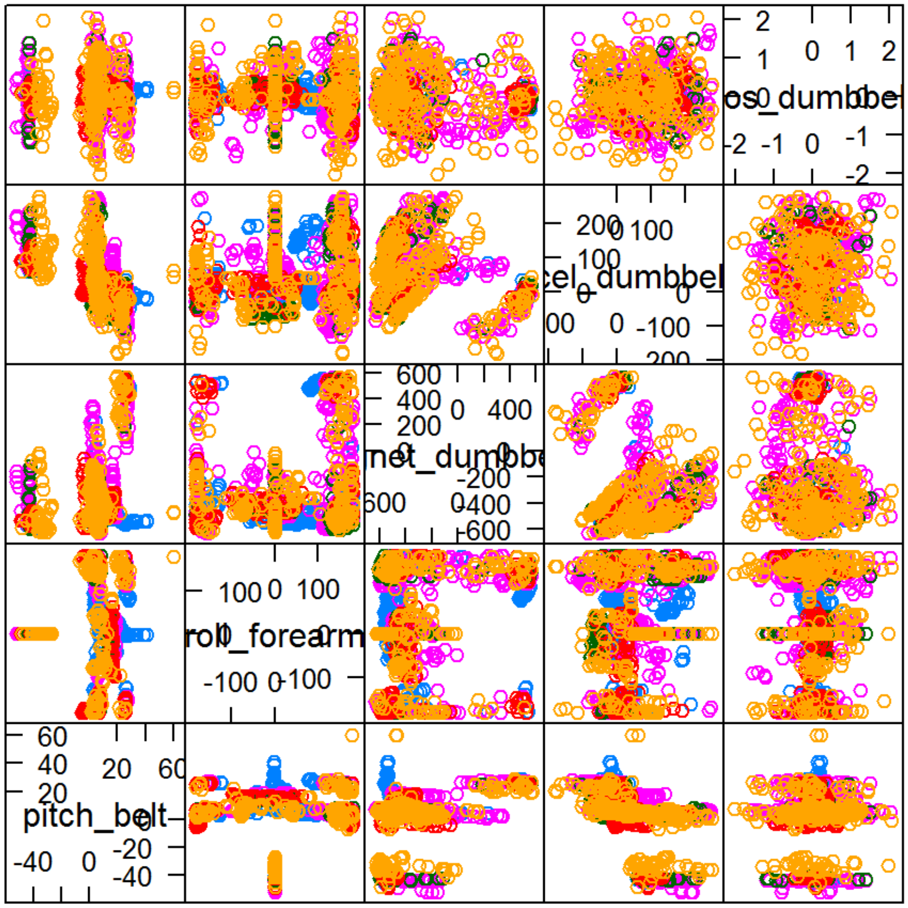
Create Variables Feature Plots

The first step to improving the algorithm is to use the subset of the 20 most influential variables to create scatter matrix plots of each subset of 5. These do not reveal any noteworthy patterns. To save computational space I am using the mini data sets as the source data.

```
featurePlot(x=mini.training[,c("roll_belt", "pitch_forearm","magnet_dumbbell_z","yaw_belt", "magnet_d
umbbell_y")],y=mini.training$classe,plot="pairs")
```

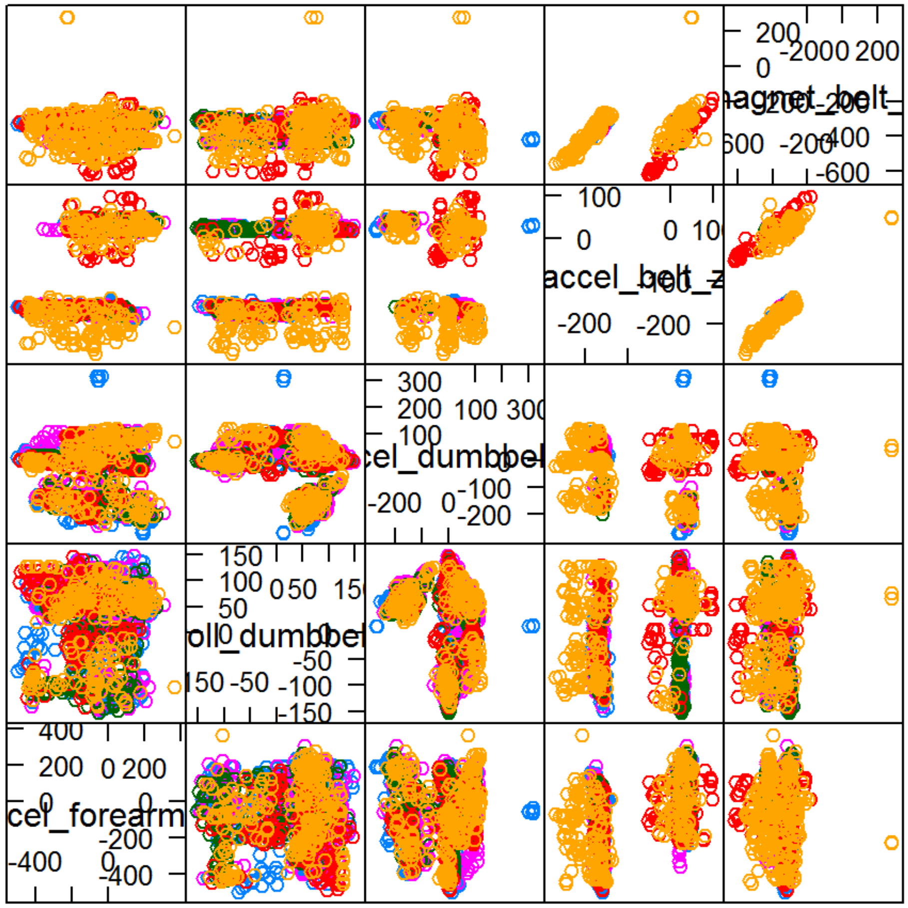



```
featurePlot(x=mini.training[,c("pitch_belt","roll_forearm","magnet_dumbbell_x","accel_dumbbell_y","gy
ros_dumbbell_y")],y=mini.training$classe,plot="pairs")
```



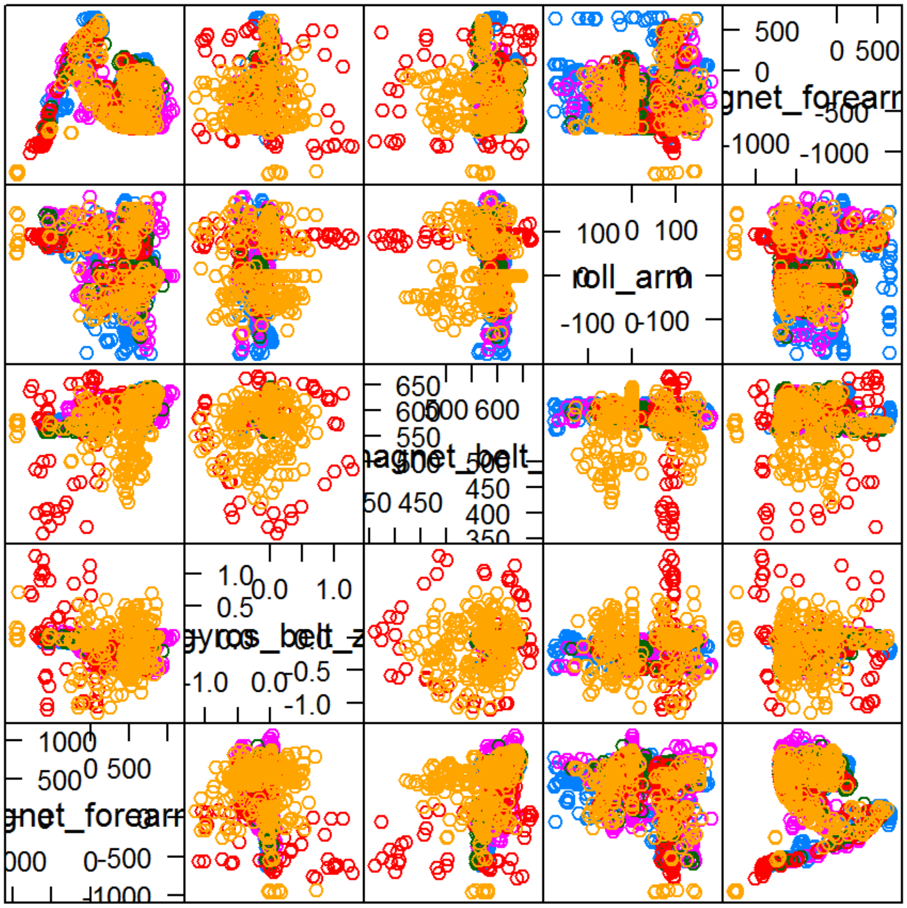
Scatter Plot Matrix

```
featurePlot(x=mini.training[,c("accel_forearm_x", "roll_dumbbell","accel_dumbbell_z","accel_belt_z","
magnet_belt_z")],y=mini.training$classe,plot="pairs")
```



Scatter Plot Matrix

```
featurePlot(x=mini.training[,c("magnet_forearm_z", "gyros_belt_z","magnet_belt_y","roll_arm","magnet_
forearm_x")],y=mini.training$classe,plot="pairs")
```



Scatter Plot Matrix

Create Models using the Top 20 most influential variables

To improve the model accuracy I will retrict the model to the top 20 variables form the anlysis above to see how this improves performance. The code below repeats uses only the top 20 variables from the data set and repeats the analysis above. Note that we are again using the mini-data set to save computation time. We are also caching the models when complete and pulling the cached models to generate this document.

The result is that restricting the variable set improves performance to an accuacy close to 93%.

Create subset of 20 most important variables

```
top.values <- c("classe","roll_belt", "pitch_forearm","magnet_dumbbell_z","yaw_belt", "magnet_dumbbel
l_y",
           "pitch_belt", "roll_forearm","magnet_dumbbell_x","accel_dumbbell_y","gyros_dumbbell_y",
           "accel_forearm_x","roll_dumbbell","accel_dumbbell_z","accel_belt_z","magnet_belt_z",
           "magnet_forearm_z","gyros_belt_z","magnet_belt_y","roll_arm","magnet_forearm_x"
            )
training.select <- training.raw[,top.values]
```

Create Test and Training Sets

```
inTrain <- createDataPartition(y=training.select$classe, p=.75, list=FALSE)
training <- training.select[inTrain,]
testing <- training.select[-inTrain,]
```

Create Mini Test and Training Sets

```
training.idx <- which(!is.na(training$classe))
testing.idx <- which(!is.na(testing$classe))
in.mini.training <- sample(training.idx, size = ceiling(length(training.idx)/10))
in.mini.testing <- sample(testing.idx, size = ceiling(length(testing.idx)/10))
mini.training <- training[in.mini.training,]
mini.testing <- testing[in.mini.testing,]
```

Create Model

```
#modFit.20.rf.mini <- train(classe ~ .,method = "rf", data=mini.training, prox=TRUE)
#save(modFit.20.rf.mini,file="modFit_20_rf_mini")
load("modFit_20_rf_mini")
pred <- predict(modFit.20.rf.mini, newdata=mini.testing)
confusionMatrix(pred,mini.testing$classe)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction   A   B   C   D   E
##          A 141   7   2   0   0
##          B   1  89   4   1   0
##          C   0   7  74   4   0
##          D   0   0   3  73   0
##          E   0   1   0   3  81
##
## Overall Statistics
##
##                Accuracy : 0.933
##                  95% CI : (0.907, 0.953)
##     No Information Rate : 0.289
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.915
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.993    0.856    0.892    0.901    1.000
## Specificity             0.974    0.984    0.973    0.993    0.990
## Pos Pred Value          0.940    0.937    0.871    0.961    0.953
## Neg Pred Value          0.997    0.962    0.978    0.981    1.000
## Prevalence              0.289    0.212    0.169    0.165    0.165
## Detection Rate          0.287    0.181    0.151    0.149    0.165
## Detection Prevalence    0.305    0.193    0.173    0.155    0.173
## Balanced Accuracy       0.984    0.920    0.932    0.947    0.995
```

```
varImp(modFit.20.rf.mini)
```

```
## rf variable importance
##
##                    Overall
## roll_belt          100.00
## magnet_dumbbell_z   59.10
## yaw_belt            57.68
## pitch_forearm       55.54
## magnet_dumbbell_y   49.79
## pitch_belt          39.00
## magnet_dumbbell_x   36.09
## roll_forearm        35.11
## magnet_belt_z       33.18
## accel_dumbbell_y    30.89
## roll_dumbbell       30.46
## accel_belt_z        26.82
## roll_arm            24.81
## magnet_belt_y       18.79
## accel_dumbbell_z    16.67
## accel_forearm_x     11.72
## gyros_dumbbell_y    10.68
## magnet_forearm_z     5.68
## gyros_belt_z         1.03
## magnet_forearm_x     0.00
```

Make Predictions

```
pred <- predict(modFit.20.rf.mini, newdata=testing.select)
pred
```

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

Create Models Using the top 10 most influential variables

We further restrict the variable set to determine if this will improve performance. The upshot is that it does not, so we take the 20 variable set to be the working variable set.

Select subset of 10 most important variables

```
top.values <- c("classe","roll_belt", "pitch_forearm","magnet_dumbbell_z","yaw_belt", "magnet_dumbbel
l_y",
           "pitch_belt", "roll_forearm","magnet_dumbbell_x","accel_dumbbell_y","gyros_dumbbell_y"
            )
training.select <- training.raw[,top.values]
```

Create Training and Test Sets

```
inTrain <- createDataPartition(y=training.select$classe, p=.75, list=FALSE)
training <- training.select[inTrain,]
testing <- training.select[-inTrain,]
```

Create Mini Training and Test Sets

```
training.idx <- which(!is.na(training$classe))
testing.idx <- which(!is.na(testing$classe))
in.mini.training <- sample(training.idx, size = ceiling(length(training.idx)/10))
in.mini.testing <- sample(testing.idx, size = ceiling(length(testing.idx)/10))
mini.training <- training[in.mini.training,]
mini.testing <- testing[in.mini.testing,]
```

Create Model

```
#modFit.10.rf.mini <- train(classe ~ .,method = "rf", data=mini.training, prox=TRUE)
#save(modFit.10.rf.mini,file="modFit_10_rf_mini")
load("modFit_10_rf_mini")
pred <- predict(modFit.10.rf.mini, newdata=mini.testing)
confusionMatrix(pred,mini.testing$classe)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   A   B   C   D   E
##          A 135   8   0   2   0
##          B   2  80   3   0   1
##          C   2   2  80   6   3
##          D   0   2   2  85   0
##          E   0   1   0   0  77
## 
## Overall Statistics
##                                          
##                Accuracy : 0.931          
##                  95% CI : (0.905, 0.952)
##     No Information Rate : 0.283          
##     P-Value [Acc > NIR] : <2e-16         
##                                          
##                   Kappa : 0.912          
##  Mcnemar's Test P-Value : NA             
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.971    0.860    0.941    0.914    0.951
## Specificity             0.972    0.985    0.968    0.990    0.998
## Pos Pred Value          0.931    0.930    0.860    0.955    0.987
## Neg Pred Value          0.988    0.968    0.987    0.980    0.990
## Prevalence              0.283    0.189    0.173    0.189    0.165
## Detection Rate          0.275    0.163    0.163    0.173    0.157
## Detection Prevalence    0.295    0.175    0.189    0.181    0.159
## Balanced Accuracy       0.971    0.923    0.955    0.952    0.974
```

```
varImp(modFit.10.rf.mini)
```

```
## rf variable importance
##
##                   Overall
## roll_belt           100.0
## yaw_belt             51.7
## magnet_dumbbell_z    41.4
## pitch_belt           38.9
## pitch_forearm        38.6
## magnet_dumbbell_y    30.9
## accel_dumbbell_y     19.9
## magnet_dumbbell_x    18.9
## roll_forearm         15.8
## gyros_dumbbell_y      0.0
```

Make Predictions

```
pred <- predict(modFit.10.rf.mini, newdata=testing.select)
pred
```

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

Increase Data Analyzed to 20% Using the 20 Variable Model to Create Final Data Set

To futher improve the accuracy of the model we expand the size of the data set. The data sets above use the mini data set which includes 10% of the data. In the analysis below we use 20% of the original data for testing and training. This improves the accuracy to approximately 97%.

Note that for random forests, the running time makes using the complete data set time prohibitive.

Select Top 20 Variables

```
top.values <- c("classe","roll_belt", "pitch_forearm","magnet_dumbbell_z","yaw_belt", "magnet_dumbbel
l_y",
           "pitch_belt", "roll_forearm","magnet_dumbbell_x","accel_dumbbell_y","gyros_dumbbell_y",
           "accel_forearm_x","roll_dumbbell","accel_dumbbell_z","accel_belt_z","magnet_belt_z",
           "magnet_forearm_z","gyros_belt_z","magnet_belt_y","roll_arm","magnet_forearm_x"
            )
training.select <- training.raw[,top.values]
```

Create Test and Training Sets

```
inTrain <- createDataPartition(y=training.select$classe, p=.75, list=FALSE)
training <- training.select[inTrain,]
testing <- training.select[-inTrain,]
```

Create Mini Test and Training Sets with 20% of the data

```
training.idx <- which(!is.na(training$classe))
testing.idx <- which(!is.na(testing$classe))
in.mini.training <- sample(training.idx, size = ceiling(length(training.idx)/1))
in.mini.testing <- sample(testing.idx, size = ceiling(length(testing.idx)/1))
mini.training <- training[in.mini.training,]
mini.testing <- testing[in.mini.testing,]
```

Create Model

```
#modFit.20.rf.full <- train(classe ~ .,method = "rf", data=mini.training, prox=TRUE)
#save(modFit.20.rf.full,file="modFit_20_rf_full")
load("modFit_20_rf_full")
pred <- predict(modFit.20.rf.full, newdata=mini.testing)
confusionMatrix(pred,mini.testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1389   31    0    6    2
##          B    2  898   35    0    6
##          C    2   16  806   17    7
##          D    2    4   14  780    9
##          E    0    0    0    1  877
##
## Overall Statistics
##
##                Accuracy : 0.969
##                  95% CI : (0.963, 0.973)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.96
##  Mcnemar's Test P-Value : 1.35e-09
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.996    0.946    0.943    0.970    0.973
## Specificity             0.989    0.989    0.990    0.993    1.000
## Pos Pred Value          0.973    0.954    0.950    0.964    0.999
## Neg Pred Value          0.998    0.987    0.988    0.994    0.994
## Prevalence              0.284    0.194    0.174    0.164    0.184
## Detection Rate          0.283    0.183    0.164    0.159    0.179
## Detection Prevalence    0.291    0.192    0.173    0.165    0.179
## Balanced Accuracy       0.992    0.968    0.966    0.982    0.987
```

```
varImp(modFit.20.rf.full)
```

```
## rf variable importance
##
##                    Overall
## roll_belt          100.00
## yaw_belt            70.51
## magnet_dumbbell_z   65.65
## pitch_forearm       64.71
## magnet_dumbbell_y   55.46
## pitch_belt          51.19
## roll_forearm        34.43
## magnet_dumbbell_x   33.53
## magnet_belt_z       31.64
## roll_dumbbell       27.42
## accel_dumbbell_y    27.31
## magnet_belt_y       23.15
## accel_dumbbell_z    17.70
## accel_belt_z        15.40
## accel_forearm_x     13.88
## roll_arm            13.04
## gyros_belt_z        10.05
## magnet_forearm_z     6.57
## magnet_forearm_x     2.77
## gyros_dumbbell_y     0.00
```

Make Predicitions Using Final Data Set to Create Final Predictions

To create the final predictions we use the final model and apply this to the test data.

```
pred <- predict(modFit.20.rf.full, newdata=testing.select)
pred
```

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