Practical Machine Learning/ Prediction Assignment

Course Assignment:- Prediction

by Geoffrey Hill

Please find below my submission for the course project.

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, 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://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Data

The training data for this project are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

The test data are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

The data for this project come from this source: http://groupware.les.inf.puc-rio.br/har.

Overview and Summary

The training data was loaded and perused. The testing data wasn't accessed until after the training had been performed. The outcome to be measured, "classe", recorded a correct exercise as "A" with other types of errors were recorded as "B" to "E". A set.seed value was used to help reproducibility. Libraries and packages were loaded as outlined below. The training set of data was further subsetted, with 60% being used for training purposes, and the remaining 40% being used for subsequent testing of our model(s). The most accurate model was selected and used to make the final prediction. Out of interest, I re-ran the test using several different methods for training, and also recorded the time taken to generate the models. The random Forest method was most accurate but also took the longest time. As a final step I plotted accuracy vs time for several different methods I tried.

Result-- sneak peak

The model generated using randomForest predicted the following results: ## [1] B A B A A E D B A A B C B A E E A B B B

Importing Data, and subsetting the training group

```
setwd("C:/@Coursera/@assignment_1")
library(doParallel)
```

```
## Warning: package 'doParallel' was built under R version 3.3.1
```

```
## Warning: package 'foreach' was built under R version 3.3.1
```

```
require(caret);require(rpart); require(rpart.plot);
```

```
## Warning: package 'caret' was built under R version 3.3.1
```

```
## Warning: package 'rpart.plot' was built under R version 3.3.1
```

```
cl<- detectCores()
registerDoParallel(cl-1)
```

The test data was downloaded from: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv, and stored locally. Next a set.seed was made to help with reproducibility,

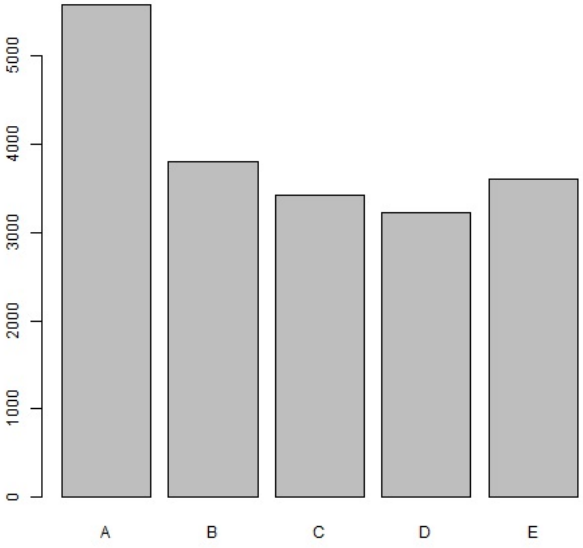
```
set.seed(13579)
atrain <- "pml-training.csv"
dtrain<- read.csv(atrain, header=TRUE,  na.strings=c("NA","#DIV/0!", ""))
trainingset<-dtrain[,colSums(is.na(dtrain)) == 0]
```

The training data was further subsetted, with 60% to be used for training models, and the remaining 40% to test the proposed models. Non-prognostic data such as (1) columns with all NA values, (2) timestamp data, and (3) other identifiers, were removed.

```
dim(dtrain)
```

```
## [1] 19622   160
```

```
plot(dtrain$classe)
```



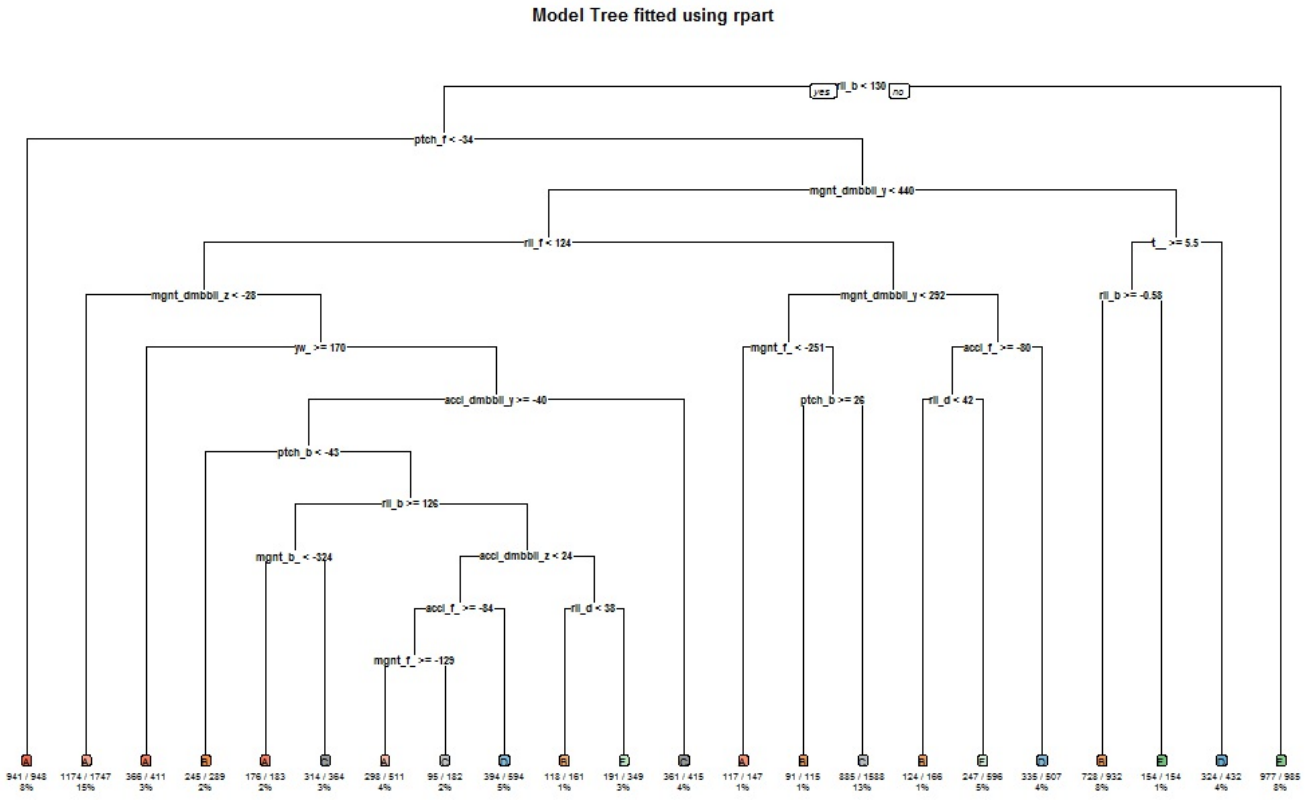

```
trainingset <- trainingset[, -c(1:7)]
trim_dtrain <- createDataPartition(y=trainingset$classe, p=0.60, list=FALSE)

TR_dtrain <- trainingset[trim_dtrain, ]
TEST_dtrain <- trainingset[-trim_dtrain, ]

dim(TEST_dtrain); dim(TR_dtrain)
```

```
## [1] 7846   53
```

```
## [1] 11776    53
```

Then the fun could begin. Firstly I made a CART type model using RPART, and then plotted a chart to try to visualise what was going on.

```
##
a1start <- proc.time()
model1 <- rpart(classe ~ ., data=TR_dtrain, method="class")
# couldn't get fancyRpart or anything else to be readable!
#rpart.plot(model1, type=2, extra=101, under=TRUE, tweak=2)
rpart.plot(model1, varlen = 3, faclen=0, cex = 0.7, type=0, extra=102, under=TRUE, gap=0, space = 0, main = "Model Tree fitted using rpart")
```



The next step is to test this model against the remaining 40% of the training data that has been "unseen" by our training. I recorded the time taken for the calculations also.

```
prediction1 <- predict(model1, TEST_dtrain, type = "class")
confusionMatrix(prediction1, TEST_dtrain$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2029  316   34  180   56
##          B   63  826   72   32   93
##          C   65  138 1084  229  177
##          D   45  129   85  701   64
##          E   30  109   93  144 1052
##
## Overall Statistics
##
##                Accuracy : 0.7255
##                  95% CI : (0.7154, 0.7353)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6507
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9091   0.5441   0.7924  0.54510   0.7295
## Specificity            0.8956   0.9589   0.9060  0.95076   0.9413
## Pos Pred Value         0.7759   0.7606   0.6403  0.68457   0.7367
## Neg Pred Value         0.9612   0.8976   0.9538  0.91425   0.9392
## Prevalence             0.2845   0.1935   0.1744  0.16391   0.1838
## Detection Rate         0.2586   0.1053   0.1382  0.08934   0.1341
## Detection Prevalence   0.3333   0.1384   0.2158  0.13051   0.1820
## Balanced Accuracy      0.9023   0.7515   0.8492  0.74793   0.8354
```

```
p1<- postResample(prediction1, TEST_dtrain$classe)
p1
```

```
##  Accuracy     Kappa
## 0.7254652 0.6506937
```

```
p1[["Accuracy"]]
```

```
## [1] 0.7254652
```

```
OutofSample = 1-p1[["Accuracy"]]

a1time <- proc.time() - a1start
a1time
```

```
##    user  system elapsed
##    1.84    0.03    1.87
```

```
a1_acc <- p1[["Accuracy"]]
```

The accuracy reported was 0.759, with out of sample errors as 0.241. So these results were OK, and quick! Next step is to use a randomForest model to see if there are improved predictions.

```
#require(randomForest); a2start<- proc.time()
#modfitA<- train(classe~.,data=TR_dtrain, method="rf",verbose=FALSE)
#a2time <- proc.time() - a2start
#a2time
```

Again, to repeat the prediction process using the TEST_ component of the original training data

```
predict2 <- predict(modfitA, TEST_dtrain) #type = "class"
```

```
## Loading required package: randomForest
```

```
## Warning: package 'randomForest' was built under R version 3.3.1
```

```
## randomForest 4.6-12
```

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

```
##
## Attaching package: 'randomForest'
```

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

```
confusionMatrix(predict2, TEST_dtrain$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2231    5    0    0    0
##          B    0 1511    4    0    0
##          C    0    2 1359    5    2
##          D    0    0    5 1281    1
##          E    1    0    0    0 1439
##
## Overall Statistics
##
##                Accuracy : 0.9968
##                  95% CI : (0.9953, 0.9979)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.996
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9996   0.9954   0.9934   0.9961   0.9979
## Specificity            0.9991   0.9994   0.9986   0.9991   0.9998
## Pos Pred Value         0.9978   0.9974   0.9934   0.9953   0.9993
## Neg Pred Value         0.9998   0.9989   0.9986   0.9992   0.9995
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1926   0.1732   0.1633   0.1834
## Detection Prevalence   0.2850   0.1931   0.1744   0.1640   0.1835
## Balanced Accuracy      0.9993   0.9974   0.9960   0.9976   0.9989
```

```
p2<- postResample(predict2, TEST_dtrain$classe)
p2
```

```
##  Accuracy     Kappa
## 0.9968137 0.9959695
```

```
p2[["Accuracy"]]
```

```
## [1] 0.9968137
```

```
OutofSample = 1-p2[["Accuracy"]]
a2_acc <- p2[["Accuracy"]]
```

Much better! The reported accuracy is 0.9924 and the Out of Sample error is 0.0076. This would be enough to go ahead with the making our predictions based on the (as yet unseen) testing data. In the next step we access that for the first time, and make our predictions.

```
atest<- "pml-testing.csv"
finaltesting <- read.csv(atest, header=TRUE)

predictfinal <- predict(modfitA, finaltesting) #, type="class"
predictfinal
```

```
##  [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
```

That could be the end of the assignment

But I did a little extra!

I won't bore you with the full code as spat out by R, but I repeated the steps above using additional models and measured accuracy, and time taken. Model1 (above) used rpart, model2 used randomForest. I repeated these steps and used: (model3) -- method = "gbm", or stochastic gradient boosting. (model4) -- method="nnet", or neural network. (model5) -- method="nb", or naive bayes. (model6) -- method="svmRadial", or Support Vector Machine with radial basis function. (model7) -- method="lvq", or learning vector quantisation.

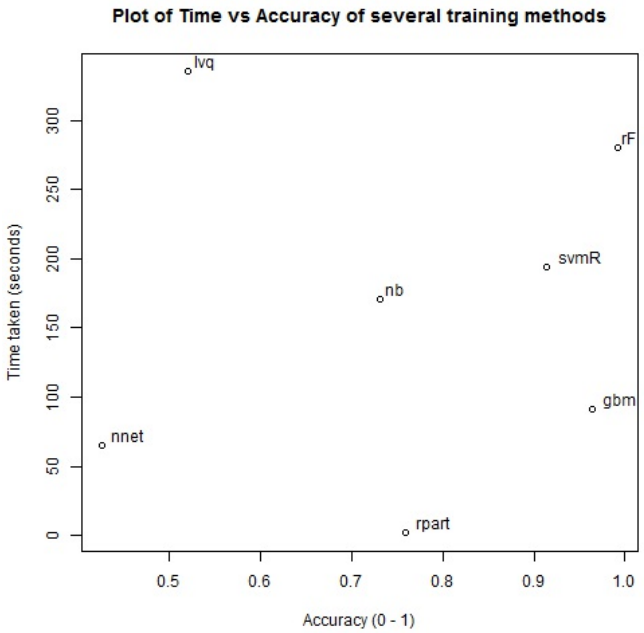
```
library("calibrate")
```

```
## Warning: package 'calibrate' was built under R version 3.3.1
```

```
## Loading required package: MASS
```

```
## now make a table, to plot
r_part<- c(a1_acc, a1time)
rF<- c(a2_acc, a2time)
gbm_<- c(a3_acc, a3time)
n_net<- c(a4_acc, a4time)
n_b<- c(a5_acc, a5time)
svmR<- c(a6_acc, a6time)
l_v_q<- c(a7_acc, a7time)

tb_final <- rbind(r_part, rF, gbm_, n_net, n_b, svmR, l_v_q)
n_tb<- c("rpart", "rF", "gbm", "nnet", "nb", "svmR", "lvq")

plot(tb_final, main="Plot of Time vs Accuracy of several training methods",
     xlab="Accuracy (0 - 1)",
     ylab="Time taken (seconds)",)
```

```
textxy(tb_final[,1], tb_final[,2], n_tb, cex = 1)
```



Interestingly, time alone seems unrelated to accuracy, however the more accurate methods did tend to take more time. Random Forest generated the best accuracy. The final prediction from the random forest model was:

```
predictfinal
```

```
##  [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
```