Machine Learning Course Project

This report attempts to predict type of exercise based on a total of 157 possible predictor variables.

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

## Warning: Failed to load RGtk2 dynamic library, attempting to install it.

## Please install GTK+ from http://r.research.att.com/libs/GTK\_2.24.17-X11.pkg  
## If the package still does not load, please ensure that GTK+ is installed and that it is on your PATH environment variable  
## IN ANY CASE, RESTART R BEFORE TRYING TO LOAD THE PACKAGE AGAIN  
## Rattle: A free graphical interface for data mining with R.  
## Version 4.0.0 Copyright (c) 2006-2015 Togaware Pty Ltd.  
## Type 'rattle()' to shake, rattle, and roll your data.  
## randomForest 4.6-12  
## Type rfNews() to see new features/changes/bug fixes.

Next, an examination of the predictor variables was made to determine which could easily be excluded from the model based on having no variation. From this we can remove 60 of the variables. Based on this information we then create a new training data set that removes these variables, leaving 97 possible predictors.

nzv <- nearZeroVar(pml.training)  
filterPML.training <-pml.training[,-nzv]  
dim(filterPML.training)

## [1] 19622 100

The first variable is useless as it is just an ID number, so is the name of the person so we want to remove these.

MyTraining <- filterPML.training[c(-1)]

MyTraining <- MyTraining[c(-1)]  
dim(MyTraining)

## [1] 19622 98

We also need to remove the NA values. We set a threshold of 50% - if more than 50% of the column includes NA values we want to remove them from the possible set of predictors.

trainingNoNA <- MyTraining

for(i in 1:length(MyTraining)) {  
 if( sum( is.na( MyTraining[, i] ) ) /nrow(MyTraining) >= .5 ) {  
 for(j in 1:length(trainingNoNA)) {  
 if( length( grep(names(MyTraining[i]), names(trainingNoNA)[j]) ) ==1) {  
 trainingNoNA <- trainingNoNA[ , -j]  
 }   
 }   
 }  
}

dim(trainingNoNA)

## [1] 19622 57

Now we have a possible 56 predictors (57-our classe variable to be predicted).

Because the testing set is reserved for the ultimate course test, we split the training data into a training and testing set that we can use.

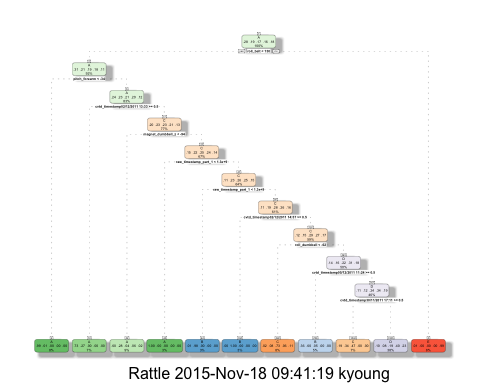
inTrain <- createDataPartition(y=trainingNoNA$classe, p=0.6, list=FALSE)  
pmlTraining <- trainingNoNA[inTrain, ]  
pmlTesting <- trainingNoNA[-inTrain, ]  
dim(pmlTraining); dim(pmlTesting)

## [1] 11776 57

## [1] 7846 57

The first model we want to examine is a Tree as this is a simple and powerful method that allows for easy interpreation.

modeltree <- train(classe~.,data=pmlTraining, method = "rpart")  
fancyRpartPlot(modeltree$finalModel)



Then we want to look at how well it predicts the testing data we created

predicttree <- predict(modeltree, pmlTesting)  
confusionMatrix(predicttree, pmlTesting$classe)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 1686 336 42 46 19  
## B 143 658 13 0 0  
## C 127 255 714 32 92  
## D 268 269 599 1208 694  
## E 8 0 0 0 637  
##   
## Overall Statistics  
##   
## Accuracy : 0.6249   
## 95% CI : (0.6141, 0.6356)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.5294   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.7554 0.43347 0.5219 0.9393 0.44175  
## Specificity 0.9211 0.97535 0.9219 0.7210 0.99875  
## Pos Pred Value 0.7919 0.80835 0.5852 0.3976 0.98760  
## Neg Pred Value 0.9045 0.87770 0.9013 0.9838 0.88821  
## Prevalence 0.2845 0.19347 0.1744 0.1639 0.18379  
## Detection Rate 0.2149 0.08386 0.0910 0.1540 0.08119  
## Detection Prevalence 0.2713 0.10375 0.1555 0.3872 0.08221  
## Balanced Accuracy 0.8382 0.70441 0.7219 0.8302 0.72025

This model is not very accurate - only 62%. We should try a different approach, so the next model we use is a random forrest model as this model tends to be more accurate than a simple tree.

modelrf <- randomForest(classe~.,data=pmlTraining)

And now for how well it predicts

predictrf<- predict(modelrf, pmlTesting)  
confusionMatrix(predictrf, pmlTesting$classe)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 2230 0 0 0 0  
## B 2 1518 2 0 0  
## C 0 0 1361 0 0  
## D 0 0 5 1286 1  
## E 0 0 0 0 1441  
##   
## Overall Statistics  
##   
## Accuracy : 0.9987   
## 95% CI : (0.9977, 0.9994)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9984   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.9991 1.0000 0.9949 1.0000 0.9993  
## Specificity 1.0000 0.9994 1.0000 0.9991 1.0000  
## Pos Pred Value 1.0000 0.9974 1.0000 0.9954 1.0000  
## Neg Pred Value 0.9996 1.0000 0.9989 1.0000 0.9998  
## Prevalence 0.2845 0.1935 0.1744 0.1639 0.1838  
## Detection Rate 0.2842 0.1935 0.1735 0.1639 0.1837  
## Detection Prevalence 0.2842 0.1940 0.1735 0.1647 0.1837  
## Balanced Accuracy 0.9996 0.9997 0.9974 0.9995 0.9997

This is much better, with an accuracy of 99%. This is therefore the model we will use to predict the provided test data for the assignment.

We need the test data to have the same columns as the training data, so next we make sure the data sets have the same variables/columns

pml.testing <- read.csv("~/Desktop/courseraR/pml-testing.csv")  
columnsused <- colnames(pmlTraining[,-57])  
ClassTesting <- pml.testing[columnsused]  
dim(ClassTesting)

## [1] 20 56

Now we predict the test data (note that I gave up getting random forrest to predict. I spent hours searching but could never get past the error "Type of predictors in the new data do not match that of the training data" so the results below are from the poorer tree model)

predictTest <- predict(modeltree, ClassTesting)  
predictTest

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

A final note on error: The out of sample error rate, or generalization error, is the error set you get on a new data set. This will usually be higher than the in sample error rate due to some overfitting. In the tree model the out of sample error was 1-0.62 = 0.38, not very good. In the random forrest model the out of sample error was 1-0.99 = 0.01, a very good error rate and thus one reason we selected this model as the final model.