Predicting if exercises are being done right

Synopsis

Developing a machine learning algorithm to predict if exercises are being done in the correct manner using human activity recognition data

Process

Let us load some required packages:

```
library(caret)
library(tree)
library(randomForest)
library(rpart)
```

Now let us load the data and take a quick look:

```
training <- read.csv("pml-training.csv", header = TRUE, na.strings = c("","NA"))
testing <- read.csv("pml-testing.csv", header = TRUE, na.strings = c("","NA"))
dim(training)
```

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

There are about 20K observations of 160 variables. Looking at the structure of the training dataframe, there are a lot of NA's and also another NA value "#DIV/0!" that's all over the data. Let us reload the data with this new NA string also included.

```
training <- read.csv("pml-training.csv", header = TRUE, na.strings = c("","NA","#DIV/0!"))
testing <- read.csv("pml-testing.csv", header = TRUE, na.strings = c("","NA", "#DIV/0!"))
```

Closer look at the data shows a number of columns with only NA values. To get rid off these columns from the dataset, lets find the proportion of NAs in each columns and remove those columns that have high proportion of NAs:

```
prop <- round((colSums(is.na(training))/19622), 2)
table(prop)
```

```
## prop
##    0 0.98    1
##   60   94    6
```

From the table above, we can see that there are about 100 columns with mostly (98%) NAs. We can safely drop these columns and our model shouldn't be affected. Let us get rid off these columns in both the training and test sets:

```
training <- training[colSums(is.na(training))/19622 < 0.97]
testing <- testing[colSums(is.na(testing))/20 < 0.97]
```

I confirmed that the columns in the new training and test sets are the same except for the response column.

Also, there are a few other columns (1 thorugh 6) that are just timestamps and other indicators, but not the actual variables that we would use in our model. Let us get rid off all these columns to reduce unneccessary computational complexity.

```
toRmv <- c(1,2,3,4,5,6)
training <- training[,-toRmv]
testing <- testing[,-toRmv]
```

Now for cross-validation purposes and to look at error rates, as we have a lot of observations in our dataset, let us partition the training dataset in to nTrain and nTest so that we can test and improve our prediction models.

```
set.seed(13383)
index <- createDataPartition(training$classe, p = 0.75, list = FALSE)
nTrain <- training[index,]
nTest <- training[-index,]
```

Since the response variable "classe" is a factor, let us use classification trees method "rpart" for fitting our model and see how that model performs:

```
modFit <- train(classe ~ ., method = "rpart", data = nTrain)
modFit$results[1,]
```

```
##           cp  Accuracy     Kappa AccuracySD    KappaSD
## 1 0.03916263 0.5143589 0.3692207 0.05462728 0.08696681
```

From the results above, we see that the expected accuracy is around 51% and so the expected error rate is 49%.

Let us predict using this model on the nTest partition and compare those predictions to actual values and see how the accuracy compares to the expected value above:

```
prediction <- predict(modFit, newdata = nTest)
cm <- confusionMatrix(prediction, nTest$classe)
cm$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   4.908238e-01   3.338678e-01   4.767392e-01   5.049194e-01   2.844617e-01
## AccuracyPValue  McnemarPValue
##  1.079649e-203            NaN
```

An accuracy rate of 49%, which is just below the expected rate from above, leaves much to be desired from the above model. So we have to look at other methods. One extension of the classification trees algorithm that is very effective is the random forests method. Lets try that method now:

```
rFit <- randomForest(classe ~ ., data = nTrain)
```

Let us look at the error rates from the above fit:

```
mean(colMeans(rFit$err.rate))
```

```
## [1] 0.004351641
```

The expected error rate of this model is 0.004. Let us see how this compares to the actual error rate after predicting using this model. Let us test our model on the test partition and compare those predictions to the actual values:

```
rfPrediction <- predict(rFit, newdata = nTest)
rfCM <- confusionMatrix(rfPrediction, nTest$classe)
rfCM$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9979608      0.9974205      0.9962531      0.9990217      0.2844617
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

Accuracy above 0.99 meaning the error rate is close to the expected value from above. Looks like this is a very good model fit for this dataset. Infact a detailed look at the confusion matrix provided in the appendix shows that this model is performing very well overall.

This was further confirmed when the predictions this model produced for the original "testing" data set turned out to be accurate also.

Appendix

Detailed Confusion Matrix of the fitted Randomforest model on the test partition from the training dataset:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    3    0    0    0
##          B    0  946    2    0    0
##          C    0    0  853    5    0
##          D    0    0    0  799    0
##          E    0    0    0    0  901
##
## Overall Statistics
##
##                Accuracy : 0.998
##                  95% CI : (0.9963, 0.999)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9974
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
```

```
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9968   0.9977   0.9938   1.0000
## Specificity            0.9991   0.9995   0.9988   1.0000   1.0000
## Pos Pred Value         0.9979   0.9979   0.9942   1.0000   1.0000
## Neg Pred Value         1.0000   0.9992   0.9995   0.9988   1.0000
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1929   0.1739   0.1629   0.1837
## Detection Prevalence   0.2851   0.1933   0.1750   0.1629   0.1837
## Balanced Accuracy      0.9996   0.9982   0.9982   0.9969   1.0000
```