Machine learning algorithm to predict quality of physical excercise

Introduction:

The goal of this project is to design an algorithm to accurately predict, using data collected from accelerometers fitted to various devices, the quality of the weight lifting exercise performed by an user wearing the device.[1]

The data from the accelerometers consists of observations recording acceleration and other attributes relating to the movement of dumbbells, belts and other items worn by the user on which the accelerometers were embedded.

Data Collection: The data was downloaded on 15 February 2015 from the following URL: https://class.coursera.org/predmachlearn-011/human_grading/view/courses/973546/assessments/4/submissions The downloaded data has been cached, so that updates do not affect our analysis.

Predictive analysis: Predictive analysis was used to:

- Construct, based on a training set, a model that helps predict the quality of the exercise
- Apply the model on a different data set (the test set) to test the accuracy of the model The R statistical package was used to perform our analysis. The caret and randomForest libraries for R were used.

Modeling: A Random Forest model trained on the training set was used to predict the quality of the exercise.[2]

```
x = read.csv("pml-training.csv", na.strings = c("", "NA"))
# changes all 'NA' and blanks in the dataset to NA

str(x)
# we can see that columns are mostly numeric, some integers and some
are
# factors, including the all-important 'classe' column

sum(complete.cases(x))
# we can see that very very few rows are complete..most have many
'NA's-
# hence, the data needs to be cleaned of the missing observations

head(x)
tail(x)
# we can observe that most of the missing data is in particular
columns
# and that there are quite a few columns which do not seem to have
any
# missing data

y = t(x)
sum(complete.cases(y))
# we see that as maany as around 60 columns of the original input
data
# have complete data. That's enough predictors to build a decent
model!

# Now, we remove those columns of the input data that do not have a
lot of
# observations
cc2 = complete.cases(y)
y = x[, cc2]
```

```
# Further, from reading the head and tail, we could identify some
columns
# as being metadata (such as timestamps). Given the nature of data
# (weightlifting excercises, we rule out the possibility of the data
being
# a time series.

# There remained a question on whether the 'user_name' column could
have
# any value as a predcitor. After all,the quality of excercise could
very
# well depend on the person doing it. The following table was run to
do a
# bit of fishing..
round(prop.table(table(y$user_name, y$classe), 1), 2)

# It was decided to keep the username column, as we are using a
tree-based
# model anyway and could rely on the model to not consider a
predictor
# with not much predictive value
```

```
y = y[, -c(1, 3:7)]
# the metadata such as timestamps mentioned above arer removed.
# 'User_name' is kept for now. The column numbers were easy enough
to spot
# from runnig head(x) and tail(x).
dim(y)

# At this point, it may be useful to save the processed data set
(this is
# just a record keeping step!)
write.csv(y, file = "y.csv", quote = FALSE)

# store the number of predictors in a separate variable for ease of
use
# (as the last column is being predicted and the rest used to
predict it,
# we know that the number of predictors is no of cols -1)
z = dim(y)[2] - 1

# Forcing all the integer fields to numeric. Leaving factors as
factors
# (the first & the last columns).
for (i in 2:z) {
    y[, i] = as.numeric(y[, i])
}
# we could also use apply functions here instead of a for-loop..

y[, z + 1] = as.factor(y[, z + 1])
# using coercion to make sure that the 'classe' is a factor. This is
not
# really a required step.
```

Cross validation:

The model was cross-validated on a test set spliced off from the training set provided. 95% of the training set was used to train the model while 5% was used as the test set.

Further, the model built was also cross validated by testing it on randomly generated chunks of 20 and 100 observations from the training set.

```
# Now that the procesing and exploration is over, we go to the model
# building part..

# Importing libraries
library(caret)
library(randomForest)

# Slice the training set data into two so that we can cross-
validate..
set.seed(9994)
trainIndex = createDataPartition(y$classe, p = 0.95, list = FALSE)
#create a 95-5 split..
training = y[trainIndex, ]
testing = y[-trainIndex, ]
dim(training)
dim(testing)
```

Choice of model: Since the data being predicted is not continuous, a tree-based classification model, rather than a regression based model was considered more appropriate.

The RandomForest method was chosen as the method generates many trees and averages its predictions across the trees, helping smooth out the randomness that a single iteration of the tree model can throw up. The RandomForest method is known to provide good accuracy in classfication problems.

```
set.seed(9)
model <- randomForest(classe ~ ., data = training, mtry = 3)
# Build a predictor model using randomForest.Having a greater
'mtry'' puts
# stress on the memory needed to run R. See ?randomForest for more
# details.
```

Out of sample error: Since the value being predicted is a set of 5 classes (i.e. not continuous), Accuracy was considered the appopriate measure of the effectiveness of the model.

The model built was tested on the test set and the results were satisfactory (Accuracy > 0.99). Based on the testing on the spliced-off test set, we expect an out of sample error rate of < 0.01 for predictions generated by the model. The in-sample accuracy of the model (i.e. the accuracy on the data on which it was trained) was 1.0. However, the low out of sample error rate allays fears of overfitting.

```
pred = predict(model, testing)
confusionMatrix(testing$classe, pred)
```

Further testing of the model: The model was also tested on randomly pulled smaller sets of observations from the input data. The results were satisfactory (Accuracy being 1.0 on most occasions, and around 0.99 on a few).

```
# Pick a sample of 20 from the input data and predict..20 was chosen
as
# the ultimate test set that had to be predicted was of size 20!
M = y[sample(dim(y)[1], 20, replace = FALSE), ]
predM = predict(model, M)
confusionMatrix(M$classe, predM)

# Do it a few more times..
M = y[sample(dim(y)[1], 20, replace = FALSE), ]
predM = predict(model, M)
confusionMatrix(M$classe, predM)

M = y[sample(dim(y)[1], 20, replace = FALSE), ]
predM = predict(model, M)
confusionMatrix(M$classe, predM)

# Similarly, try a few runs with random sample of 100.. test it for
'M' of
# size 100:
M = y[sample(dim(y)[1], 100, replace = FALSE), ]
predM = predict(model, M)
confusionMatrix(M$classe, predM)

# On the last few observations of the set..(just because the tail is
# easier to observe!)
predM = predict(model, y[19601:19622, ])
confusionMatrix(y$classe[19601:19622], predM)

# On some observations from the middle of the data..
predM = predict(model, y[2000:2869, ])
confusionMatrix(y$classe[2000:2869], predM)

# THESE TESTS ABOVE ARE NOT REALLY NECESSARY BUT WERE DOEN JUST TO
SEE THE
# MODEL RUN!
```

Result: Finally, the model was run on the final test set of 20 observations for which the class of exercise were not known and had to be predicted by the model. The model predicted all the 20 observations accurately (verified by the grader scripts at the URL: https://class.coursera.org/predmachlearn-011/assignment), in line with the expectation of an accuracy of 0.99 (formed on the basis of the prediction on the internal test set of 5%). The error rate on the 20 observations was 0.0. However, given a larger set, error rates of ~0.01 are reasonable to expect.

```
t = read.csv("pml-testing.csv", na.strings = c("", "NA"))
str(t)
head(t)
# Processing of this set was not considered necessary as the model
used
# predictors that were already complete (without missing data) and
were in
# numeric/integer and factor format. No errors were expected if the
model
# tried to coerce integers to numeric and none were got!

predANS = predict(model, t)  #Finally, the predictions

# Since the project required generation of different files for each
# observation, a few files were written to the working directory..
for (i in 1:20) {
    write.table(as.character(predANS[i]), file = paste(i, ".txt"),
quote = FALSE,
        row.names = FALSE, col.names = FALSE)
}
```

References: 1. Description of the weight lifting exercise. http://groupware.les.inf.puc-rio.br
/har#weight_lifting_exercises 2. Random Forest: http://en.wikipedia.org/wiki/Random_forest