Practical Machine Learning: Course programming Assignment

Background:

This is a programming assignment of the course “Practical Machine Learning” by the John Hopkins University.

Note: Text shaded below in grey references the corresponding code in R.

Objective:

The data provided is a collection of data gathered from participants doing exercises. The goal of the program is to predict the manner in which the participants did the exercises. This is the "classe" variable in the data set.

Approach:

1. **Download data sets**
2. **Cleaning the data**
3. **Splitting the data into a training and test set**
4. **Plotting and reviewing the data**
5. **Building a model**
6. **Testing the model**
7. **Predicting the results**

1. Download the data sets

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

I assume the files downloaded and in the same directory as the code. Load the data, create test and training set (don’t use the testing data provided! This is used for prediction later).

```
trainRawDat <- read.csv("pml-training.csv")

testRawDat <- read.csv("pml-testing.csv")
```

A quick peek at the data shows that we have 19622 observations in 160 variables. Note that the last variable “classe” identifies the problem_id (Letter A to E).

Unfortunately, the some variables set also contain a large number of “NA’s”. Time for cleaning.

2. Cleaning the data

Check for each variable the number and percent of NA’s: we see that there are a large number of variables that have >97% of NA’s. We are going to eliminate those variables by creating a True/False vector representing those variables we want in (TRUE) or out (FALSE) of the data set and reducing the raw data accordingly:

```
goodVAR<- pctNA<0.97
```

```
cleanDat<-trainRawDat[,goodVAR]
```

Upon review of the data, I realized that the first 6 variables do not contain useful data for building a prediction model (name, timestamp, ….). I eliminate them.

The corresponding variables are also excluded from the test set (the results we want to predict). And the data sets are converted to the right data format for further processing and modeling.

3. Splitting the training data

I chose to split the data 85/15 into training and testing data:

Note that the test set is the cross validation set, a subset of the training set.

```
inTrain <- createDataPartition(y=cleanDat$classe, p=0.85, list=FALSE)
```

```
training <- cleanDat[inTrain,]
```

```
testing <- cleanDat[-inTrain,]
```

It is NOT the data we need to predict later on (this is called test4real).

4. Plotting and reviewing the data

(not required, but I wanted to play with the various variables and plotting features a bit ☺)

5. Building the model

I used two models to see which ones provided the better results: Random forest and GBM. This step took me quite some time. Key learnings included: finding the “do parallel” mode to speed up the calculation; using a pre-processing (“pca”) option, etc. The modeling still takes considerable time. (Note to self: check further options to improve speed).

```
registerDoParallel()
```

```
fitControl <- trainControl(method = "repeatedcv", number = 2, repeats = 2)
```

```
modelFitRF <- train(classe ~ ., data = training, method = "rf", preProcess="pca", trControl =
fitControl)
```

```
modelFitGBM<- train(classe ~ ., data = training, method = "gbm", preProcess="pca", trControl =
fitControl)
```

6. Testing the Model

Comparing the results of the RF vs the GBM model shows that RF produces better/more accurate results (code example for RF, same for GBM):

```
print(modelFitRF)

predTrain <- predict(modelFitRF,training)

table(predTrain, training$classe)

predTest <- predict(modelFitRF,testing)

table(predTest, testing$classe)

conMatRF<- confusionMatrix(testing$classe, predict(modelFitRF, testing))

print(conMatRF)
```

Results RF (extract r-console print):

```
> print(conMatRF)
Confusion Matrix and Statistics
          Reference
Prediction   A   B   C   D   E
         A 827   0   5   5   0
         B   7 558   3   0   1
         C   1   3 507   2   0
         D   0   1  13 468   0
         E   0   2   1   0 538

Overall Statistics
               Accuracy : 0.985
                 95% CI : (0.98, 0.9891)
    No Information Rate : 0.2838
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9811
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: A Class: B Class: C Class: D Class: E
Sensitivity            0.9904   0.9894   0.9584   0.9853   0.9981
Specificity            0.9953   0.9954   0.9975   0.9943   0.9988
Pos Pred Value         0.9881   0.9807   0.9883   0.9710   0.9945
Neg Pred Value         0.9962   0.9975   0.9909   0.9972   0.9996
Prevalence             0.2838   0.1917   0.1798   0.1615   0.1832
```

```
Detection Rate        0.2811   0.1897   0.1723   0.1591   0.1829
Detection Prevalence  0.2845   0.1934   0.1744   0.1638   0.1839
Balanced Accuracy     0.9928   0.9924   0.9780   0.9898   0.9984
```

Results GBM:

```
> print(conMatGBM)
Confusion Matrix and Statistics
          Reference
Prediction   A   B   C   D   E
         A 743  16  39  34   5
         B  47 437  54  16  15
         C  25  37 432  13   6
         D  13  30  48 382   9
         E  12  33  25  23 448

Overall Statistics
               Accuracy : 0.83
                 95% CI : (0.816, 0.8435)
    No Information Rate : 0.2855
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7851
 Mcnemar's Test P-Value : 1.535e-13

Statistics by Class:

                     Class: A Class: B Class: C Class: D Class: E
Sensitivity            0.8845   0.7902   0.7224   0.8162   0.9275
Specificity            0.9553   0.9447   0.9654   0.9596   0.9622
Pos Pred Value         0.8877   0.7680   0.8421   0.7925   0.8281
Neg Pred Value         0.9539   0.9511   0.9317   0.9650   0.9854
Prevalence             0.2855   0.1880   0.2033   0.1591   0.1642
Detection Rate         0.2525   0.1485   0.1468   0.1298   0.1523
Detection Prevalence   0.2845   0.1934   0.1744   0.1638   0.1839
Balanced Accuracy      0.9199   0.8675   0.8439   0.8879   0.9449
```

As seen above, the RF model is producing better/more accurate results (98.5%). I'm thus using RF to predict on the test/prediction data set.

7. Predict Results

This step is very straightforward:

```
resultsRF<- as.character(predict(modelFitRF,test4real))
```

Finally, using the code recommended by the course website to write the results into *.txt files.

```
pml_write_files = function(x){
        n = length(x)
        for(i in 1:n){
        filename = paste0("problem_id_",i,".txt")
        write.table(x[i],file=filename,quote=FALSE,row.names=FALSE,col.names=FALSE)
        }
}
pml_write_files(resultsRF)
```

Done ☺