Practical Machine Learning: Course Project

1. Project Description

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. If you use the document you create for this class for any purpose please cite them as they have been very generous in allowing their data to be used for this kind of assignment.

What you should submit

The goal of your project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

---

2. Loading the Data

First we will load the libraries required for this project:

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(randomForest)
```

```
## 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
```

```
library(rpart)
library(rattle)
```

```
## 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

## IN ANY CASE, RESTART R BEFORE TRYING TO LOAD THE PACKAGE AGAIN

## Rattle: A free graphical interface for data mining with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

Next we will download the data files and read them into memory:

```
trainingUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

if (!file.exists('pml-training.csv')){
    download.file(trainingUrl, dest='pml-training.csv', method='curl')
}
if (!file.exists('pml-testing.csv')){
    download.file(trainingUrl, dest='pml-testing.csv', method='curl')
}

trainingData <- read.csv('pml-training.csv', na.strings=c('NA', '', '#DIV/0!'))
testingData <- read.csv('pml-testing.csv', na.strings=c('NA', '', '#DIV/0!'))
```

```
dim(trainingData)
```

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

```
dim(testingData)
```

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

3. Partitioning the Training Data

To perform cross-validation, we will split the training data into two sets using the `createDataPartition()` function:

- a training set (`train`), consisting of 60% of the training data, that will be used to train out models; and
- a test set (`test`), consisting of the remaining 40% of the training data, that will be used for preliminary testing during model selection.

We will not use the test data we loaded above until the end of the project; we will call this the `testFinal` data set.

```
# Split the training set into training and test sets:
inTrain <- createDataPartition(y=trainingData$classe, p=0.6, list=FALSE)
train <- trainingData[inTrain, ]
```

```
test <- trainingData[-inTrain, ]
# make a copy of the testing data that will be cleaned and tested
# after model selection and training
testFinal <- testingData
# check that the dimensions make sense: same number of columns
dim(train)
```

```
## [1] 11776   160
```

```
dim(test)
```

```
## [1] 7846  160
```

```
dim(testFinal)
```

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

4. Data Cleaning

To clean the data, we will:

1. Remove first 6 columns of obviously irrelevant parameters (person's name, id number, etc.);
2. Remove parameters with low variance, which tend to have little predictive value;
3. Remove parameters with $> 50\%$ NA values

We will apply these cleaning methods to each of our 3 data sets:

```
# remove first 6 columns of obviously irrelevant parameters (person's name, id number, etc.)
train <- train[, -(1:6)]
test <- test[, -(1:6)]
testFinal <- testFinal[, -(1:6)]
```

```
# remove parameters with low variance:
lowVariance <- nearZeroVar(train, saveMetrics=TRUE)
train <- train[, !lowVariance$nzv]
test <- test[, !lowVariance$nzv]
testFinal <- testFinal[, !lowVariance$nzv]
```

```
# define a function to identify fraction of NAs in column; delete columns with > 50% NAs
fractionNAs <- apply(train, 2, function(col) sum(is.na(col)))/nrow(train)
train <- train[!(fractionNAs > 0.5)]
test <- test[!(fractionNAs > 0.5)]
testFinal <- testFinal[!(fractionNAs > 0.5)]
```

```
# check the new dimensions
dim(train)
```

```
## [1] 11776    54
```

```
dim(test)
```

```
## [1] 7846   54
```

```
dim(testFinal)
```

```
## [1] 19622    54
```

4. Model Selection

We will train two models using the `train` partition of the and training data, and will test them using the `test` partition of the training data. We will select the model with the higher accuracy.

First, we will try a decision tree using `rpart`:

```
# produce a decision tree
rpartModel <- rpart(classe ~ ., data=train, method='class')
#fancyRpartPlot(rpartModel)
```

```
# test it on the test partition and get accuracy using confusionMatrix
rpartPredictions <- predict(rpartModel, test, type='class')
confusionMatrix(rpartPredictions, test$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2032  313   39   91  104
##          B   77  830   55   91  137
##          C   12  119 1097  171   95
##          D   82  109   98  787   71
##          E   29  147   79  146 1035
##
## Overall Statistics
##
##                Accuracy : 0.7368
##                  95% CI : (0.7269, 0.7465)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6653
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9104   0.5468   0.8019   0.6120   0.7178
## Specificity            0.9026   0.9431   0.9387   0.9451   0.9374
## Pos Pred Value         0.7879   0.6975   0.7343   0.6861   0.7208
## Neg Pred Value         0.9620   0.8966   0.9573   0.9255   0.9365
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2590   0.1058   0.1398   0.1003   0.1319
## Detection Prevalence   0.3287   0.1517   0.1904   0.1462   0.1830
## Balanced Accuracy      0.9065   0.7449   0.8703   0.7785   0.8276
```

Next, we will do a random forest classifier:

```
rfModel <- randomForest(classe ~ ., data=train, do.trace=FALSE)
rfPredictions <- predict(rfModel, test, type='class')
confusionMatrix(rfPredictions, test$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
```

```
##          A 2232    1    0    0    0
##          B    0 1517    5    0    0
##          C    0    0 1363   16    0
##          D    0    0    0 1269    3
##          E    0    0    0    1 1439
##
## Overall Statistics
##
##                Accuracy : 0.9967
##                  95% CI : (0.9951, 0.9978)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9958
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9993   0.9963   0.9868   0.9979
## Specificity            0.9998   0.9992   0.9975   0.9995   0.9998
## Pos Pred Value         0.9996   0.9967   0.9884   0.9976   0.9993
## Neg Pred Value         1.0000   0.9998   0.9992   0.9974   0.9995
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1933   0.1737   0.1617   0.1834
## Detection Prevalence   0.2846   0.1940   0.1758   0.1621   0.1835
## Balanced Accuracy      0.9999   0.9993   0.9969   0.9932   0.9989
```

The random forest method gives (by far) superior results. We will select this model and use it on the training data.

5. Predictions for the Test Data

The accuracy of the random forest model was 99.6% (with a 95% confidence interval of 0.994 to 0.997). The out-of-sample error rate is therefore estimated to be 1 - 0.996 = 0.004 = 0.4%. When applied to the 20 samples in the test data set, we can expect approximately 0 missclassifications.

```
finalPredictions <- predict(rfModel, testFinal, method='class')
head(finalPredictions)
```

```
## 1 2 3 4 5 6
## A A A A A A
## Levels: A B C D E
```