Practical Machine Learning Course Porject

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).

Abstract

The goal of this project is to predict the manner in which they did the exercise. This report addresses on the following questions:

- how to built the model;
- how to use cross validation;
- what is the expected out of sample error;
- why you made the choices you did.

Finally, the prediction model is used to predict 20 different test cases.

Data

The training data for this project are downloaded from the link as below:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

The test data are downloaded from the link as below:

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.

```
## [1] "C:/Jing/ProgramCode/DataScience/C8_Practical_Machine_Learning"
```

```
## Loading required package: lattice
## Loading required package: ggplot2
## Rattle: A free graphical interface for data mining with R.
## XXXX 3.1.0 Copyright (c) 2006-2014 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
## Loading required package: rpart
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```r
# Download data.
# url_raw_training <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
file_dest_training <- "pml-training.csv"
# download.file(url=url_raw_training, destfile=file_dest_training)
# url_raw_testing <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
file_dest_testing <- "pml-testing.csv"
# download.file(url=url_raw_testing, destfile=file_dest_testing)

# Import the data treating empty values as NA.
df_training <- read.csv(file_dest_training, na.strings=c("NA",""), header=TRUE)
colnames_train <- colnames(df_training)
df_testing <- read.csv(file_dest_testing, na.strings=c("NA",""), header=TRUE)
colnames_test <- colnames(df_testing)

# Verify that the column names (excluding classe and problem_id) are identical in the training and test
all.equal(colnames_train[1:length(colnames_train)-1], colnames_test[1:length(colnames_train)-1])
```

```
## [1] TRUE
```

Data Pre-process

There are some columns containing the NA or missing values. These did not contribute well to the prediction. We chose a feature set that only included complete columns.

```r
# Count the number of non-NAs in each col.
nonNAs <- function(x) {
    as.vector(apply(x, 2, function(x) length(which(!is.na(x)))))
}

# Build vector of missing data or NA columns to drop.
colcnts <- nonNAs(df_training)
drops <- c()
for (cnt in 1:length(colcnts)) {
    if (colcnts[cnt] < nrow(df_training)) {
        drops <- c(drops, colnames_train[cnt])
    }
}

# Drop NA data and the first 7 columns as they're unnecessary for predicting.
df_training <- df_training[,!(names(df_training) %in% drops)]
df_training <- df_training[,8:length(colnames(df_training))]

df_testing <- df_testing[,!(names(df_testing) %in% drops)]
df_testing <- df_testing[,8:length(colnames(df_testing))]
```

Algorithms

We were provided with a large training set (19,622 entries) and a small testing set (20 entries). Instead of performing the algorithm on the entire training set, as it would be time consuming and wouldn't allow for an attempt on a testing set, I chose to divide the given training set into four roughly equal sets, each of which was then split into a training set (comprising 60% of the entries) and a testing set (comprising 40% of the entries).

```
# Divide the given training set into 4 roughly equal sets.
set.seed(914)
ids_small <- createDataPartition(y=df_training$classe, p=0.25, list=FALSE)
df_small1 <- df_training[ids_small,]
df_remainder <- df_training[-ids_small,]
set.seed(914)
ids_small <- createDataPartition(y=df_remainder$classe, p=0.33, list=FALSE)
df_small2 <- df_remainder[ids_small,]
df_remainder <- df_remainder[-ids_small,]
set.seed(914)
ids_small <- createDataPartition(y=df_remainder$classe, p=0.5, list=FALSE)
df_small3 <- df_remainder[ids_small,]
df_small4 <- df_remainder[-ids_small,]
# Divide each of these 4 sets into training (60%) and test (40%) sets.
set.seed(914)
inTrain <- createDataPartition(y=df_small1$classe, p=0.6, list=FALSE)
df_small_training1 <- df_small1[inTrain,]
df_small_testing1 <- df_small1[-inTrain,]
set.seed(914)
inTrain <- createDataPartition(y=df_small2$classe, p=0.6, list=FALSE)
df_small_training2 <- df_small2[inTrain,]
df_small_testing2 <- df_small2[-inTrain,]
set.seed(914)
inTrain <- createDataPartition(y=df_small3$classe, p=0.6, list=FALSE)
df_small_training3 <- df_small3[inTrain,]
df_small_testing3 <- df_small3[-inTrain,]
set.seed(914)
inTrain <- createDataPartition(y=df_small4$classe, p=0.6, list=FALSE)
df_small_training4 <- df_small4[inTrain,]
df_small_testing4 <- df_small4[-inTrain,]
```

We made use of caret and randomForest model for prediction on the test datasets. Firstly, we train the first subtraining set 1 with both preprocessing and cross validation.

```
# Train on training set 1 of 4 with both preprocessing and cross validation.
set.seed(914)
modFit <- train(df_small_training1$classe ~ ., method="rf", preProcess=c("center", "scale"), trControl=
print(modFit, digits=3)
```

```
## Random Forest
##
## 2946 samples
##   52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered, scaled
## Resampling: Cross-Validated (4 fold)
##
## Summary of sample sizes: 2209, 2208, 2212, 2209
##
```

```
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
##    2    0.944     0.930  0.0137       0.0174
##   27    0.952     0.940  0.0149       0.0189
##   52    0.948     0.935  0.0185       0.0234
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

```
# Run against testing set 1 of 4.
predictions <- predict(modFit, newdata=df_small_testing1)
print(confusionMatrix(predictions, df_small_testing1$classe), digits=4)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 555  12   0   0   0
##          B   1 358  11   0   1
##          C   1   9 325  15   2
##          D   1   0   5 306   5
##          E   0   1   1   0 352
##
## Overall Statistics
##
##                Accuracy : 0.9669
##                  95% CI : (0.9579, 0.9743)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9581
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9946   0.9421   0.9503   0.9533   0.9778
## Specificity            0.9914   0.9918   0.9833   0.9933   0.9988
## Pos Pred Value         0.9788   0.9650   0.9233   0.9653   0.9944
## Neg Pred Value         0.9978   0.9862   0.9894   0.9909   0.9950
## Prevalence             0.2845   0.1938   0.1744   0.1637   0.1836
## Detection Rate         0.2830   0.1826   0.1657   0.1560   0.1795
## Detection Prevalence   0.2891   0.1892   0.1795   0.1617   0.1805
## Balanced Accuracy      0.9930   0.9669   0.9668   0.9733   0.9883
```

```
# Run against 20 testing set.
print(predict(modFit, newdata=df_testing))
```

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

We applied both preprocessing and cross validation on other three subdatasets.

```r
# Train on training set 2 of 4 with only cross validation.
set.seed(914)
modFit <- train(df_small_training2$classe ~ ., method="rf", preProcess=c("center", "scale"), trControl=
print(modFit, digits=3)
```

```
## Random Forest
##
## 2917 samples
##   52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered, scaled
## Resampling: Cross-Validated (4 fold)
##
## Summary of sample sizes: 2187, 2189, 2187, 2188
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
##    2    0.948     0.934  0.00844      0.01070
##   27    0.953     0.941  0.00300      0.00383
##   52    0.948     0.934  0.00309      0.00387
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

```r
# Run against testing set 2 of 4.
predictions <- predict(modFit, newdata=df_small_testing2)
print(confusionMatrix(predictions, df_small_testing2$classe), digits=4)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 547  12   0   0   0
##          B   3 354  10   6   5
##          C   0   9 320   9   0
##          D   2   0   7 301   5
##          E   0   1   1   2 347
##
## Overall Statistics
##
##                Accuracy : 0.9629
##                  95% CI : (0.9535, 0.9709)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9531
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9909   0.9415   0.9467   0.9465   0.9720
## Specificity            0.9914   0.9847   0.9888   0.9914   0.9975
## Pos Pred Value         0.9785   0.9365   0.9467   0.9556   0.9886
## Neg Pred Value         0.9964   0.9859   0.9888   0.9895   0.9937
## Prevalence             0.2844   0.1937   0.1741   0.1638   0.1839
## Detection Rate         0.2818   0.1824   0.1649   0.1551   0.1788
## Detection Prevalence   0.2880   0.1947   0.1741   0.1623   0.1808
## Balanced Accuracy      0.9912   0.9631   0.9678   0.9690   0.9847
```

```
# Run against 20 testing set provided by Professor Leek.
print(predict(modFit, newdata=df_testing))
```

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

```
# Train on training set 3 of 4 with only cross validation.
set.seed(914)
modFit <- train(df_small_training3$classe ~ ., method="rf", preProcess=c("center", "scale"), trControl=
print(modFit, digits=3)
```

```
## Random Forest
##
## 2960 samples
##   52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered, scaled
## Resampling: Cross-Validated (4 fold)
##
## Summary of sample sizes: 2220, 2221, 2219, 2220
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
##    2    0.949     0.935  0.00884      0.01118
##   27    0.956     0.944  0.00623      0.00788
##   52    0.953     0.941  0.00876      0.01108
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

```
# Run against testing set 3 of 4.
predictions <- predict(modFit, newdata=df_small_testing3)
print(confusionMatrix(predictions, df_small_testing3$classe), digits=4)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 556   9   1   2   1
##          B   0 365  14   0   4
##          C   2   6 324   8   4
##          D   2   1   5 312   3
##          E   0   0   0   1 350
##
## Overall Statistics
##
##                Accuracy : 0.968
##                  95% CI : (0.9593, 0.9753)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9595
##  Mcnemar's Test P-Value : 0.007023
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9929   0.9580   0.9419   0.9659   0.9669
## Specificity            0.9908   0.9887   0.9877   0.9933   0.9994
## Pos Pred Value         0.9772   0.9530   0.9419   0.9659   0.9972
## Neg Pred Value         0.9971   0.9899   0.9877   0.9933   0.9926
## Prevalence             0.2843   0.1934   0.1746   0.1640   0.1838
## Detection Rate         0.2822   0.1853   0.1645   0.1584   0.1777
## Detection Prevalence   0.2888   0.1944   0.1746   0.1640   0.1782
## Balanced Accuracy      0.9918   0.9733   0.9648   0.9796   0.9831
```

```
# Run against 20 testing set provided by Professor Leek.
print(predict(modFit, newdata=df_testing))
```

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

```
# Train on training set 4 of 4 with only cross validation.
set.seed(914)
modFit <- train(df_small_training4$classe ~ ., method="rf", preProcess=c("center", "scale"), trControl=
print(modFit, digits=3)
```

```
## Random Forest
##
## 2958 samples
##   52 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered, scaled
## Resampling: Cross-Validated (4 fold)
##
## Summary of sample sizes: 2217, 2219, 2219, 2219
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
##    2    0.942     0.926  0.01070      0.01361
##   27    0.948     0.935  0.00780      0.00992
##   52    0.945     0.930  0.00355      0.00452
```

```
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

```r
# Run against testing set 4 of 4.
predictions <- predict(modFit, newdata=df_small_testing4)
print(confusionMatrix(predictions, df_small_testing4$classe), digits=4)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   A   B   C   D   E
##          A 557  13   0   0   1
##          B   1 355   9   0   3
##          C   1  13 328  14   3
##          D   1   0   6 306   3
##          E   0   0   0   3 352
##
## Overall Statistics
##
##                Accuracy : 0.9639
##                  95% CI : (0.9547, 0.9717)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9544
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9946   0.9318   0.9563   0.9474   0.9724
## Specificity            0.9901   0.9918   0.9809   0.9939   0.9981
## Pos Pred Value         0.9755   0.9647   0.9136   0.9684   0.9915
## Neg Pred Value         0.9979   0.9838   0.9907   0.9897   0.9938
## Prevalence             0.2844   0.1935   0.1742   0.1640   0.1838
## Detection Rate         0.2829   0.1803   0.1666   0.1554   0.1788
## Detection Prevalence   0.2900   0.1869   0.1823   0.1605   0.1803
## Balanced Accuracy      0.9924   0.9618   0.9686   0.9706   0.9853
```

```r
# Run against 20 testing set provided by Professor Leek.
print(predict(modFit, newdata=df_testing))
```

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

Out of Sample Error

The out of sample error, the error rate after running the predict() function on the 4 testing sets, are listed:

Random Forest (preprocessing and cross validation) Testing Set 1: 1 - 0.9648 = 0.0352 Random Forest (preprocessing and cross validation) Testing Set 2: 1 - 0.9629 = 0.0371 Random Forest (preprocessing and cross validation) Testing Set 3: 1 - 0.968 = 0.032 Random Forest (preprocessing and cross validation) Testing Set 4: 1 - 0.9639 = 0.0361

Since each testing set is roughly of equal size, I decided to average the out of sample error rates derived by applying the random forest method with both preprocessing and cross validation against test sets 1-4 yielding a predicted out of sample rate of 0.0351.

CONCLUSION

I received three separate predictions by appling the 4 models against the actual 20 item training set:

1) Accuracy Rate 0.0352 Predictions: B A B A A E D B A A B C B A E E A B B B
2) Accuracy Rates 0.0371 Predictions: B A B A A E D D A A B C B A E E A B B B
3) Accuracy Rate 0.032 Predictions: B A A A A E D B A A B C B A E E A B B B
4) Accuracy Rate 0.0361 Predictions: B A B A A E D D A A B C B A E E A B B B

Finally, I choose the same two result as the final prediction: B A B A A E D B A A B C B A E E A B B B.