This file is for the course project of Practical Machine Learning - Coursera

Tilte: Machine Learning for Prediction of the Activity Types

Overview:

In this project, we analyze how well the people do a particular activity. In particular, we train machine learning models to predict the quality of an activity. The training data is from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. We then take a testing data of 20 samples and attempt to determine the quality of each sample (by deciding the type of the activity: A, B, C, D, or E). The data is available from: http://groupware.les.inf.puc-rio.br/har

==

Part 1: Load and Process the Training Data

we first load the training data to a data frame.

```
library(caret)
```

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

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

Using the "head" function, we observe that the first 7 columns have unecessary data. We thus delete them from the training data.

```
#### Drop the first 7 columns
trainingAll <- trainingRaw[, -c(1:7)]
```

Furthermore, there are several columns which have many NA and NULL values. We also delete them from the data set. The result is a clean data set "trainingAll"

```
#### Select only columns without any NA
trainingAll <- trainingAll[, colSums(is.na(trainingAll)) == 0]
#### Delete the columns without any value
trainingAll <- trainingAll[, colSums(trainingAll == "") == 0]
#### Select the data without NA
trainingAll$classe <- as.factor(trainingRaw$classe)
```

==

Part 2: Data Pre-Processing

To train and then predict our machine learning model's accuracy, we split the data set into three:

- training: to train simple learning model
- crosVal: to blend different simple learning model
- testing: to predict our models' accuracy. Since we employ "crosVal"" to blend the models, we have to use a different data set for our test.

```
set.seed(1234)
inTrain <- createDataPartition(y = trainingAll$classe, p = 0.5, list = FALSE)
training <- trainingAll[inTrain, ]
testAndCross <- trainingAll[-inTrain, ]

inTest <- createDataPartition(y = testAndCross$classe, p = 0.5, list = FALSE)
testing <- testAndCross[inTest,]
crosVal <- testAndCross[-inTest,]
```

Now, we pre-process the "trainng"" data by centering and scaling it. We then apply this pre-processing to "crosVal" and "testing". Note that to correctly evaluate our model's accuracy, "crosVal" and "testing" have to be pre-processed by the parameters obtained from "training".

```
#### Pre-process the predictor data
preObj <- preProcess(training[, -53], method = c("center", "scale"))
trainPre <- predict(preObj, training[, -53])
testPre <- predict(preObj, testing[, -53])
crosValPre <- predict(preObj, crosVal[, -53])

#### Add the classe columns
trainPre$classe <- as.factor(training$classe)
testPre$classe <- as.factor(testing$classe)
crosValPre$classe <- as.factor(crosVal$classe)
```

==

Part 3: Train Simple Machine Learning Model

There are three classification models that we tried in this work: Rpart, GBM, and RandomForest. ADA is another classification type but only available for data with two factor levels.

The training data set is "trainPre". For randomForest, we directly call the function to reduce the running time.

```
#### Tree Rpart
modRPa <- train(classe ~ ., method = "rpart", data = trainPre)
```

```
## Loading required package: rpart
```

```
#### Boosting with GBM
modGBM <- train(classe ~ ., method = "gbm", data = trainPre)
```

```
## Loading required package: gbm
## Loading required package: survival
##
```

```
## 1 1.6094 nan 0.1000 0.2331
## 2 1.4628 nan 0.1000 0.1609
## 3 1.3592 nan 0.1000 0.1306
## 4 1.2760 nan 0.1000 0.1029
## 5 1.2111 nan 0.1000 0.0854
## 6 1.1567 nan 0.1000 0.0789
## 7 1.1068 nan 0.1000 0.0644
## 8 1.0649 nan 0.1000 0.0611
## 9 1.0264 nan 0.1000 0.0615
## 10 0.9887 nan 0.1000 0.0548
## 20 0.7552 nan 0.1000 0.0217
## 40 0.5292 nan 0.1000 0.0088
## 60 0.4068 nan 0.1000 0.0055
## 80 0.3241 nan 0.1000 0.0045
## 100 0.2640 nan 0.1000 0.0021
## 120 0.2210 nan 0.1000 0.0026
## 140 0.1882 nan 0.1000 0.0014
## 150 0.1749 nan 0.1000 0.0013
```

```
#### Random Forest
library(randomForest)
```

```
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
```

```
modRFo <- randomForest(x = trainPre[, -53], y = trainPre$classe, prox = TRUE)
```

==

Part 4: Blending the Learning Models

To blend different models, we first evaluate them using the "crosValPre" and then blend them together with "crosValPre$classe". Note that we only blend GBM and RF since Rpart yeilds a far worse accuracy (to be shown later). Therefore, blending Rpart will not help much.

```
predCrosGBM <- predict(modGBM, crosValPre[, -53])
predCrosRFo <- predict(modRFo, crosValPre[, -53])

#### Blend the learning models
predRFGBM <- data.frame(x1 = as.factor(predCrosGBM), x2 = as.factor(predCrosRFo), classe = crosValPre$cl
combModRFGBM <- train(classe ~ ., method = "gbm", data = predRFGBM)
```

```
## Iter TrainDeviance ValidDeviance StepSize Improve
## 1 1.6094 nan 0.1000 0.5637
## 2 1.2596 nan 0.1000 0.3637
## 3 1.0372 nan 0.1000 0.2684
## 4 0.8733 nan 0.1000 0.2146
## 5 0.7441 nan 0.1000 0.1718
## 6 0.6412 nan 0.1000 0.1423
## 7 0.5565 nan 0.1000 0.1188
## 8 0.4860 nan 0.1000 0.0982
```

```
##      7        0.4976             nan     0.1000    0.1206
##      8        0.4258             nan     0.1000    0.1002
##      9        0.3658             nan     0.1000    0.0827
##     10        0.3158             nan     0.1000    0.0691
##     20        0.0981             nan     0.1000    0.0121
##     40        0.0451             nan     0.1000    0.0004
##     60        0.0400             nan     0.1000   -0.0001
##     80        0.0388             nan     0.1000   -0.0003
##    100        0.0384             nan     0.1000   -0.0002
##    120        0.0384             nan     0.1000   -0.0002
##    140        0.0380             nan     0.1000   -0.0006
##    150        0.0380             nan     0.1000   -0.0012
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.6607
##      2        1.2087             nan     0.1000    0.4236
##      3        0.9569             nan     0.1000    0.3056
##      4        0.7733             nan     0.1000    0.2370
##      5        0.6317             nan     0.1000    0.1835
##      6        0.5210             nan     0.1000    0.1457
##      7        0.4337             nan     0.1000    0.1171
##      8        0.3632             nan     0.1000    0.0949
##      9        0.3061             nan     0.1000    0.0778
##     10        0.2589             nan     0.1000    0.0631
##     20        0.0742             nan     0.1000    0.0091
##     40        0.0407             nan     0.1000   -0.0003
##     60        0.0385             nan     0.1000   -0.0004
##     80        0.0380             nan     0.1000   -0.0011
##    100        0.0379             nan     0.1000   -0.0006
##    120        0.0377             nan     0.1000   -0.0005
##    140        0.0377             nan     0.1000   -0.0010
##    150        0.0377             nan     0.1000   -0.0005
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.5678
##      2        1.2584             nan     0.1000    0.3711
##      3        1.0355             nan     0.1000    0.2723
##      4        0.8708             nan     0.1000    0.2105
##      5        0.7441             nan     0.1000    0.1707
##      6        0.6419             nan     0.1000    0.1408
##      7        0.5567             nan     0.1000    0.1176
##      8        0.4857             nan     0.1000    0.0973
##      9        0.4267             nan     0.1000    0.0835
##     10        0.3773             nan     0.1000    0.0709
##     20        0.1423             nan     0.1000    0.0154
##     40        0.0658             nan     0.1000    0.0013
##     50        0.0571             nan     0.1000    0.0005
```

==

Part 5: Predicting the Accuracies of Our Learning Models

We now use "testPre" to predict the accuracies of our models. For GBM and RF

```r
predTestRPa <- predict(modRPa, testPre[, -53])
predTestGBM <- predict(modGBM, testPre[, -53])
predTestRFo <- predict(modRFo, testPre[, -53])
```

For the blended GBM+RF

```r
predTestRFGBM <- data.frame(x1 = as.factor(predTestGBM), x2 = as.factor(predTestRFo))
combPredTestRFGBM <- predict(combModRFGBM, predTestRFGBM)
```

The predicted accuracies are

```r
tmp1 <- confusionMatrix(predTestRPa, testPre$classe)$overall[1]
tmp2 <- confusionMatrix(predTestGBM, testPre$classe)$overall[1]
tmp3 <- confusionMatrix(predTestRFo, testPre$classe)$overall[1]
tmp4 <- confusionMatrix(combPredTestRFGBM, testPre$classe)$overall[1]

accuracyTable <- data.frame(Rpart = tmp1, GBM = tmp2, RandomForest = tmp3, GBMnRF = tmp4)
accuracyTable
```

```
##              Rpart       GBM RandomForest   GBMnRF
## Accuracy 0.4922544 0.9637179     0.993885 0.993885
```

We observe that Rpart is far worse than GBM and RF. Also, the predictions of GBM and RF alone are alone so accurate. The blending thus has little effect.

==

Part 6: Testing Our Learning Models to the Real Testing Data

We now use our models to determine the quality of the activities given in testing set. We first load the testing data. We then select only the columns that used to train the models

```r
trueTestingRaw  <- read.csv("pml-testing.csv")
trueTestingAll <- trueTestingRaw[names(trainingAll[,-53])]
```

Now we pre-process the testind data using the parameters obtained from the training data.

```r
#### Pre-process the test data
trueTestingAll <- predict(preObj, trueTestingAll)
```

We then determine the activity type with our 3 models: GBM, RF, and GBm+RF

```r
#### GBM and RandomForest
predTrueGBM <- predict(modGBM, trueTestingAll)
predTrueRFo <- predict(modRFo, newdata = trueTestingAll)

#### Combining predictors
predTrueRFGBM <- data.frame(x1 = as.factor(predTrueGBM),
                            x2 = as.factor(predTrueRFo))
combPredTrueRFGBM <- predict(combModRFGBM, predTrueRFGBM)
```

The results are

```
predTrueGBM
```

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

```
predTrueRFo
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  B  A  B  A  A  E  D  B  A  A  B  C  B  A  E  E  A  B  B  B
## Levels: A B C D E
```

```
combPredTrueRFGBM
```

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

We observe that the three sets of activity types are identical. This is due to the fact that the accuracies of our 3 models: GBM, RandomForest, and GBM+RF are very high.