Practical Machine Learning Final Project

by Jian Shi

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, we will use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants to predict the manner in which they did the exercise.

Preparing

```r
library(caret)
```

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

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

```r
library(rpart)
library(rpart.plot)
library(RColorBrewer)
library(rattle)
```

```
## 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.
```

```r
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
```

```r
library(knitr)
library(corrplot)
```

Get the Data

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

training <- read.csv(url(trainUrl), na.strings=c("NA","#DIV/0!",""))
testing <- read.csv(url(testUrl), na.strings=c("NA","#DIV/0!",""))
dim(training); dim(testing)
```

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

```
## [1]  20 160
```

The training data set contains 19622 observations and 160 variables, while the testing data set contains 20 observations and 160 variables. The “classe” variable in the training set is the outcome to predict.

Clean the data

In this step, we will clean the data and get rid of observations with missing values as well as some meaningless variables. First, we remove the first column as it is just an index.

```r
training = training[,-1]
testing = testing[,-1]
```

Next, we remove columns with NAs.

```r
training <- training[, colSums(is.na(training)) == 0]
testing <- testing[, colSums(is.na(testing)) == 0]
```

Then, we remove some columns that are not contributing much to accelerometer measurements. These columns contains keywords such as “window” and “timestamp”.

```r
classe <- training$classe
trainRemove <- grepl("^X|timestamp|window", names(training))
training <- training[, !trainRemove]
training <- training[, sapply(training, is.numeric)]
training$classe <- classe

testRemove <- grepl("^X|timestamp|window", names(testing))
testing <- testing[, !testRemove]
testing <- testing[, sapply(testing, is.numeric)]
dim(training); dim(testing)
```

```
## [1] 19622    53
```

```
## [1] 20 53
```

The cleaned training data set contains 19622 observations and 53 variables, while the testing data set contains 20 observations and 53 variables. The “classe” variable is still in the cleaned training set. ### Slice the data Now we can split the cleaned training set into a pure training data set (70%) and a validation data set (30%). We will use the validation data set to conduct **cross validation** in future steps.

```r
set.seed(667788)
inTrain <- createDataPartition(training$classe, p=0.70, list=F)
trainData <- training[inTrain, ]
testData <- training[-inTrain, ]
```

Data Modeling

We first see how **decision tree** looks like.

```r
dt <- rpart(classe ~ ., data=trainData, method="class")
pred1 <- predict(dt, testData, type = "class")
confusionMatrix(pred1, testData$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1381  240   20   82   40
##          B   40  561   45   23   51
##          C   42  101  775  138  126
##          D  170  189  123  664  168
##          E   41   48   63   57  697
##
## Overall Statistics
##
##                Accuracy : 0.6929
##                  95% CI : (0.681, 0.7047)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6119
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8250  0.49254   0.7554   0.6888   0.6442
## Specificity            0.9093  0.96650   0.9162   0.8679   0.9565
## Pos Pred Value         0.7833  0.77917   0.6557   0.5053   0.7693
## Neg Pred Value         0.9289  0.88809   0.9466   0.9344   0.9227
## Prevalence             0.2845  0.19354   0.1743   0.1638   0.1839
## Detection Rate         0.2347  0.09533   0.1317   0.1128   0.1184
## Detection Prevalence   0.2996  0.12234   0.2008   0.2233   0.1540
## Balanced Accuracy      0.8671  0.72952   0.8358   0.7784   0.8003
```

We can see that the accuracy is just about 70%. Now we try **random forest**. We will use **4-fold cross validation** when applying the algorithm.

```r
controlRf <- trainControl(method="cv", 4)
modelRf <- train(classe ~ ., data=trainData, method="rf", trControl=controlRf, ntree=250)
```

```
pred2 <- predict(modelRf, testData)
confusionMatrix(pred2, testData$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1672   13    0    0    0
##          B    0 1124    5    0    1
##          C    1    1 1019    7    0
##          D    0    1    2  956    3
##          E    1    0    0    1 1078
##
## Overall Statistics
##
##                Accuracy : 0.9939
##                  95% CI : (0.9915, 0.9957)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9923
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9988   0.9868   0.9932   0.9917   0.9963
## Specificity            0.9969   0.9987   0.9981   0.9988   0.9996
## Pos Pred Value         0.9923   0.9947   0.9912   0.9938   0.9981
## Neg Pred Value         0.9995   0.9968   0.9986   0.9984   0.9992
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2841   0.1910   0.1732   0.1624   0.1832
## Detection Prevalence   0.2863   0.1920   0.1747   0.1635   0.1835
## Balanced Accuracy      0.9979   0.9928   0.9957   0.9952   0.9979
```

The estimated accuracy of the model is 99.39% and the estimated out-of-sample error is 0.61%. Random Forest is expected to perform well here because it automatically selects important variables and is robust to correlated covariates & outliers in general. We are going to use this model to predict the test data.

Predict on testing data

```
result <- predict(modelRf, testing[, -dim(testing)[2]]) #the last column is problem_id
result
```

```
##  [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
```

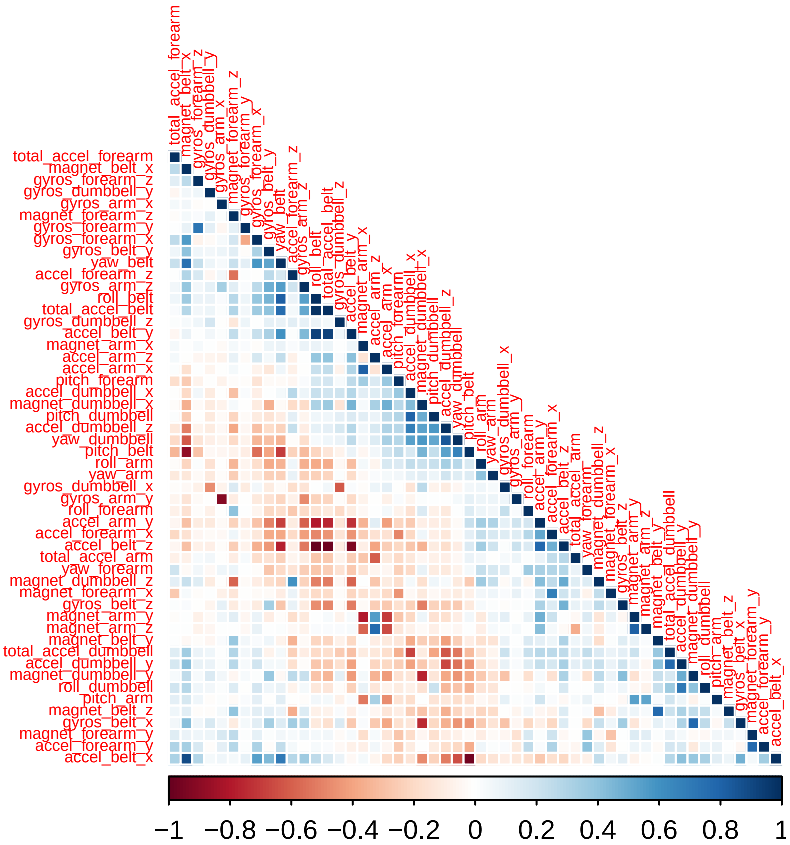
Finally we write the results to a txt file for submission.

```
x = data.frame(seq(1,20),result)
colnames(x) = c('Problem_id','Predicted result')
write.table(x,file='prediction.txt',row.names=F,col.names=T)
```

Appendix: Figures

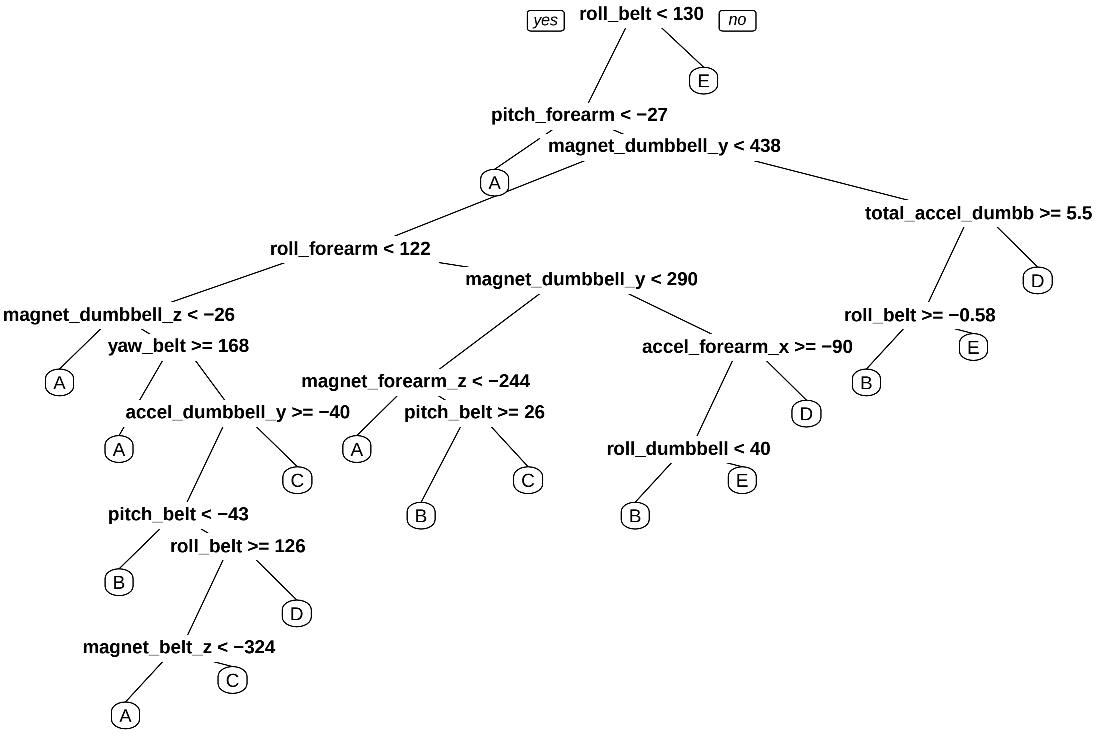
1. Correlation Matrix

```
corrPlot <- cor(trainData[, -length(names(trainData))])
corrplot(corrPlot, method="color", type='lower', order='AOE', tl.cex=0.5)
```



2. Decision Tree (we already calculate it above and saved it as “dt”)

```
prp(dt) #here I chose the fast plot over the fancy plot
```


yes
roll_belt < 130
no
E
pitch_forearm < −27
A
magnet_dumbbell_y < 438
total_accel_dumbb >= 5.5
D
roll_forearm < 122
magnet_dumbbell_y < 290
roll_belt >= −0.58
B
E
magnet_dumbbell_z < −26
A
yaw_belt >= 168
A
accel_dumbbell_y >= −40
C
pitch_belt < −43
B
roll_belt >= 126
D
magnet_belt_z < −324
A
C
magnet_forearm_z < −244
A
pitch_belt >= 26
B
C
accel_forearm_x >= −90
D
roll_dumbbell < 40
B
E