Practical Machine Learning Project

Hang Zhou

Thursday, March 19, 2015

Synopsis

Recently Wearable technology becomes a hot topic, Human Activity Recongnition (HAR) is a big contributor to this new technology. With devices like Jawbone Up, Nike Fuelband and Fitbit's help, large amount of personal activity data can be collected with low cost. How to utilitize thise data in HAR becomes a new challenge. This project tries to do analysis on the data set that published by Groupware@LES (mailto:Groupware@LES), and eventually answer the question of "Can we predict how an activity being perfomed?". If the answer is Yes, how well we can predict it.

Data Source

The data used in this project is Weight Lifting Exercises Dataset downloaded from following research project:

Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013.

Read more: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)

Introduction

In this project I will start with data cleaning, remove unnecessary data, find out columns that really matter to the analysis. Then I will apply multiple prdiction methods to the data, compare the results, and eventually find out the best one for these data.

Load Data and Clean Data

```
library(caret)
```

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

```
library(rattle)
```

```
## Rattle: A free graphical interface for data mining with R.
## Version 3.4.1 Copyright (c) 2006-2014 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(rpart)
library(rpart.plot)
raw_train <- read.csv("pml-training.csv", na.strings=c("NA",""), header=TRUE)
raw_test <- read.csv("pml-testing.csv", na.strings=c("NA",""), header=TRUE)
```

Noticed that there are lots of blank or NA fields, need some extra work to evaluate the quantity and how much that will impact our analysis.

```
fn_No_of_NonNA <- function(x) {as.vector(apply(x, 2, function(x) length(which(!is.na
(x)))))}
NoofNNA <- fn_No_of_NonNA(raw_train)
CheckOne <- data.frame(names(raw_train), NoofNNA / nrow(raw_train))
names(CheckOne) <- c("ColName", "Pct")
head(CheckOne, 20)
```

```
##                 ColName     Pct
## 1                     X 1.00000
## 2             user_name 1.00000
## 3  raw_timestamp_part_1 1.00000
## 4  raw_timestamp_part_2 1.00000
## 5        cvtd_timestamp 1.00000
## 6            new_window 1.00000
## 7            num_window 1.00000
## 8             roll_belt 1.00000
## 9            pitch_belt 1.00000
## 10             yaw_belt 1.00000
## 11     total_accel_belt 1.00000
## 12   kurtosis_roll_belt 0.02069
## 13  kurtosis_picth_belt 0.02069
## 14    kurtosis_yaw_belt 0.02069
## 15   skewness_roll_belt 0.02069
## 16 skewness_roll_belt.1 0.02069
## 17    skewness_yaw_belt 0.02069
## 18        max_roll_belt 0.02069
## 19       max_picth_belt 0.02069
## 20         max_yaw_belt 0.02069
```

The result in CheckOne shows that some columns have value for every record, while some columns only have value for less than 2.1% records. The exact same situation happens to the raw test data set as well. These columns with lots of NAs will be removed, to simplify the analysis.

```
CheckTwo <- CheckOne[which(CheckOne$Pct < 1), ]
raw_train <- raw_train[, !(names(raw_train) %in% CheckTwo$ColName )]
raw_test <- raw_test[, !(names(raw_test) %in% CheckTwo$ColName )]
##head(raw_train)
```

The first several columns are only for recording and documentation purpose, do not really contribute to the analysis, so we will remove those columns from the dataset before analysis.

```
raw_train <- raw_train[, 8: length(names(raw_train))]
raw_test <- raw_test[, 8: length(names(raw_test))]
```

Now the data is ready for us to perform further analysis. As suggested from the class, we separate 75% of the original training data for training purpose to generate the model.

```
set.seed(1234)
inTrain <- createDataPartition(y = raw_train$classe, p = 0.75, list = FALSE)
training <- raw_train[inTrain, ]
testing <- raw_train[-inTrain, ]
```

Classification Tree

There is no evidence which prediction method is the best fit so far, so I will start with Classification Tree.

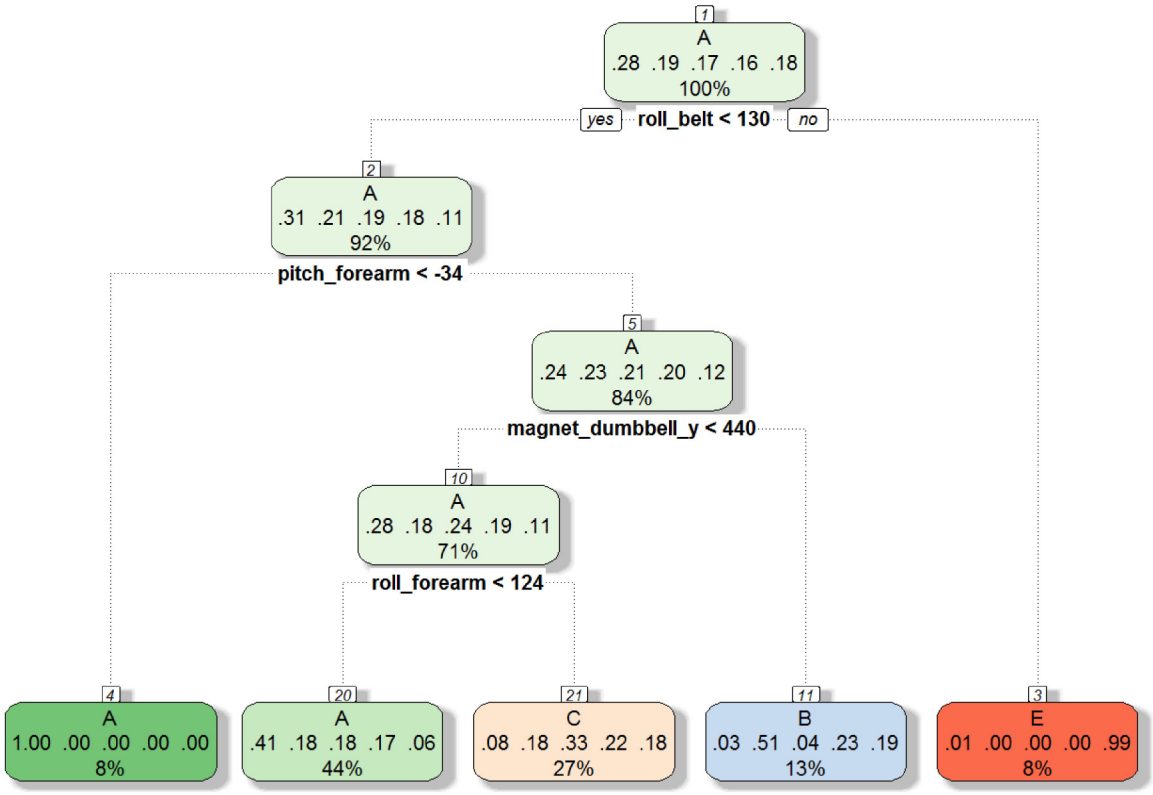
```
modFit <- train(training$classe ~ ., data = training, method = "rpart")
print(modFit)
```

```
## CART
##
## 14718 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
##
## Summary of sample sizes: 14718, 14718, 14718, 14718, 14718, 14718, ...
##
## Resampling results across tuning parameters:
##
##   cp       Accuracy  Kappa    Accuracy SD  Kappa SD
##   0.03446  0.5208    0.38177  0.04314      0.06795
##   0.05988  0.4004    0.18424  0.05983      0.09797
##   0.11535  0.3350    0.07856  0.04003      0.06019
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.03446.
```

```
print(modFit$finalModel)
```

```
## n= 14718
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 14718 10530 A (0.28 0.19 0.17 0.16 0.18)
##    2) roll_belt< 130.5 13483  9308 A (0.31 0.21 0.19 0.18 0.11)
##      4) pitch_forearm< -34.35 1163     4 A (1 0.0034 0 0 0) *
##      5) pitch_forearm>=-34.35 12320  9304 A (0.24 0.23 0.21 0.2 0.12)
##       10) magnet_dumbbell_y< 439.5 10462  7499 A (0.28 0.18 0.24 0.19 0.11)
##         20) roll_forearm< 123.5 6480  3837 A (0.41 0.18 0.18 0.17 0.06) *
##         21) roll_forearm>=123.5 3982  2663 C (0.08 0.18 0.33 0.22 0.18) *
##       11) magnet_dumbbell_y>=439.5 1858   912 B (0.029 0.51 0.041 0.23 0.19) *
##    3) roll_belt>=130.5 1235    10 E (0.0081 0 0 0 0.99) *
```

```
fancyRpartPlot(modFit$finalModel)
```



Rattle 2015-Mar-22 11:58:49 czhoujj

The accuracy of the final model that was pick is about 52.04%, that means the expected error could be up to 48%. With this model, we will use it against ther small data set (25% of the original training data set) that sets aside to verify.

```
predict1 <- predict(modFit, testing)
print(confusionMatrix(predict1, testing$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1275  390  416  356  134
##          B   28  340   31  141  131
##          C   88  219  408  307  230
##          D    0    0    0    0    0
##          E    4    0    0    0  406
##
## Overall Statistics
##
##                Accuracy : 0.495
##                  95% CI : (0.481, 0.509)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.34
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.914   0.3583   0.4772    0.000   0.4506
## Specificity             0.631   0.9163   0.7916    1.000   0.9990
## Pos Pred Value          0.496   0.5067   0.3259      NaN   0.9902
## Neg Pred Value          0.949   0.8561   0.8776    0.836   0.8899
## Prevalence              0.284   0.1935   0.1743    0.164   0.1837
## Detection Rate          0.260   0.0693   0.0832    0.000   0.0828
## Detection Prevalence    0.524   0.1368   0.2553    0.000   0.0836
## Balanced Accuracy       0.772   0.6373   0.6344    0.500   0.7248
```

The result shows that when testing with our small testing set, the accuracy dropped from 52.08% to 49.53%. The out of sample error is 50.47%.
Use this model to test again orginal 20 record test set.

```
predict2 <- predict(modFit, newdata = raw_test)
print(predict2)
```

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

Random Forest

Considering the accuracy is fair low with Classification Tree, I will try with Random Forest again since one of the Pros of Random Forest is accuracy.

```
#set.seed(1234)
modFit1 <- train(training$classe ~ ., method="rf", trControl=trainControl(method =
"cv", number = 4), data=training)
```

```
## Loading required package: randomForest
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
print(modFit1)
```

```
## Random Forest
##
## 14718 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
##
## Summary of sample sizes: 11038, 11040, 11037, 11039
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa   Accuracy SD  Kappa SD
##    2    0.9906    0.9881  0.0008986    0.001136
##   27    0.9913    0.9890  0.0018971    0.002401
##   52    0.9855    0.9816  0.0033041    0.004181
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

```
predict3 <- predict(modFit1, testing)
print(confusionMatrix(predict3, testing$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    7    0    0    0
##          B    0  939    8    1    0
##          C    0    3  845    8    1
##          D    0    0    2  794    0
##          E    0    0    0    1  900
##
## Overall Statistics
##
##                Accuracy : 0.994
##                  95% CI : (0.991, 0.996)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.992
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             1.000    0.989    0.988    0.988    0.999
## Specificity             0.998    0.998    0.997    1.000    1.000
## Pos Pred Value          0.995    0.991    0.986    0.997    0.999
## Neg Pred Value          1.000    0.997    0.998    0.998    1.000
## Prevalence              0.284    0.194    0.174    0.164    0.184
## Detection Rate          0.284    0.191    0.172    0.162    0.184
## Detection Prevalence    0.286    0.193    0.175    0.162    0.184
## Balanced Accuracy       0.999    0.994    0.993    0.994    0.999
```

```
predict4 <- predict(modFit1, newdata = raw_test)
print(predict4)
```

```
##  [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
```

The accuracy of the model is about 99.13% with cross validation, when the model is applied to the small test data set, the accuracy increased to 99.33%, which means the out of smaple error is only about 0.67%.

Conclusion

Compare the accuracy from confusionMatrix of both prediction methods, Random Forest's 99.33% definitely is much higher than Classification Tree's 49.53%. So I will use Random Forest with its prediction.

Classification Tree's prediction is:

```
print(predict2)
```

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

Random Forest's prediction is:

```
print(predict4)
```

```
##  [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
```

As class material mentioned, one of the Cons of Random Forest is Speed, indeed it took a really long time to train the model against with 75% of the training data set provided. There is another option that only use a small portion of that training data set instead of 75%, of course the time to spend on the training the model dropped, but the accuracy of the final result also dropped about 3-5%, depends on the data amount we used. If applied this calculation in some real-time processing, we might consider using less training data to make it faster, but for this project, I will choose to use the whole data that was provided.