Practical Machine Learning - Project

Hailu Teju

November 4, 2017

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, the goal is 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. The five ways are: Class A - exactly according to the specification, Class B - throwing the elbows to the front, Class C - lifting the dumbbell only halfway, Class D - lowering the dumbbell only halfway, and Class E - throwing the hips to the front. Only Class A corresponds to correct performance. More information is available from the website here: http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Data Processing

Getting the data

We first load the required R packages that will be needed for analysis and then download the training and testing data sets from the given URLs.

```r
# Load the required packages
library(caret); library(rattle); library(rpart)
library(rpart.plot); library(RColorBrewer); library(randomForest)
```

```r
trainUrl <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testUrl <- "https://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 dataset has 19622 observations and 160 variables, and the testing dataset contains 20 observations and 160 variables. We are trying to predict the outcome of the variable `classe` in the training set.

Cleaning the data

We now remove the first seven columns (predictors) of the training and testing dataset since these variables have little predicting power for the outcome `classe`.

```r
training <- training[,-c(1:7)]
testing <- testing[,-c(1:7)]
```

We also remove predictors that contain any missing values.

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

dim(training); dim(testing)
```

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

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

Our cleaned training dataset now contains 19622 observations and 53 variables and the cleaned testing data contains 20 observations and 53 variables. The first 52 variables are the same for the training and testing datasets and they have different last variables, which are `classe` for training and `problem_id` for testing datasets.

Partition the data

In order to get out-of-sample errors, we partition the cleaned training dataset into a training set (`myTraining`, 70%) for prediction and a validation set (`myTesting`, 30%) to compute the out-of-sample errors.

```r
set.seed(43876)

inTrain <- createDataPartition(y=training$classe, p=0.7,list=FALSE)

myTraining <- training[inTrain,]
myTesting <- training[-inTrain,]

dim(myTraining);dim(myTesting)
```

```
## [1] 13737    53
```

```
## [1] 5885   53
```

ML Prediction Algorithms

We use classification trees and random forests to predict the outcome.

Classification tree

In practice, `k=5 or k=10` when doing a `k-fold` cross validation. Here we use the default setting in trainControl function - which is 10.

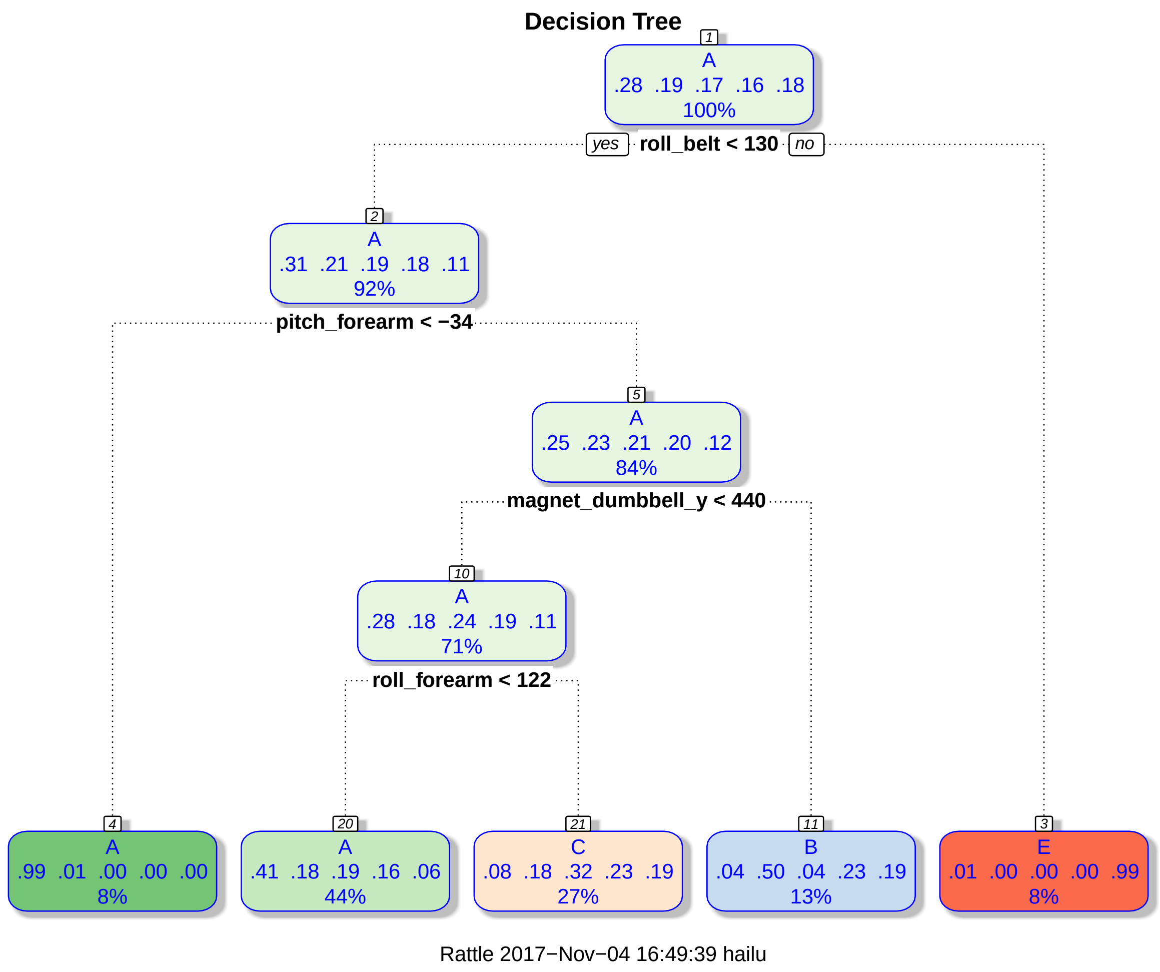
```r
control <- trainControl(method="cv")

fit_rpart <- train(classe ~. , data=myTraining, method='rpart', trControl = control)

print(fit_rpart, digits=5)
```

```
## CART
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
```

```
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 12364, 12362, 12363, 12364, 12361, 12364, ...
## Resampling results across tuning parameters:
##
##   cp        Accuracy  Kappa
##   0.034381  0.51597   0.376631
##   0.058115  0.40096   0.184461
##   0.112908  0.32190   0.057261
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.034381.
```

Viewing the decision tree with fancy:

```
fancyRpartPlot(fit_rpart$finalModel,
               main='Decision Tree',cex=1,col='blue')
```



```
# Predict outcomes using the testing set
pred_rpart <- predict(fit_rpart, myTesting)

# Show prediction result
(confM_rpart <- confusionMatrix(myTesting$classe, pred_rpart))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1549   17  106    0    2
##          B  470  401  268    0    0
##          C  465   30  531    0    0
##          D  442  157  365    0    0
##          E  156  141  276    0  509
##
## Overall Statistics
##
##                Accuracy : 0.5081
##                  95% CI : (0.4952, 0.5209)
##     No Information Rate : 0.5237
##     P-Value [Acc > NIR] : 0.9921
##
##                   Kappa : 0.3567
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.5026  0.53753  0.34347       NA  0.99609
## Specificity            0.9554  0.85639  0.88592   0.8362  0.89338
## Pos Pred Value         0.9253  0.35206  0.51754       NA  0.47043
## Neg Pred Value         0.6360  0.92731  0.79111       NA  0.99958
## Prevalence             0.5237  0.12676  0.26270   0.0000  0.08683
## Detection Rate         0.2632  0.06814  0.09023   0.0000  0.08649
## Detection Prevalence   0.2845  0.19354  0.17434   0.1638  0.18386
## Balanced Accuracy      0.7290  0.69696  0.61469       NA  0.94473
```

```
(accur_rpart <- confM_rpart$overall[1])
```

```
##  Accuracy
## 0.5080714
```

This shows that we got only a 0.5 accuracy, which means that using classification tree does not predict the outcome `classe` very well.

Random forests

```
fit_rf <- randomForest(classe ~ ., data=myTraining)

print(fit_rf, digits=5)
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = myTraining)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.5%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3901    3    0    1    1 0.001280082
## B   13 2639    6    0    0 0.007148232
## C    0   17 2379    0    0 0.007095159
## D    0    0   20 2231    1 0.009325044
## E    0    0    3    4 2518 0.002772277
```

```
# Predict outcomes using testing set
pred_rf <- predict(fit_rf, myTesting)

# Show prediction result
(confM_rf <- confusionMatrix(myTesting$classe, pred_rf))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    0    0    0    0
##          B    6 1130    3    0    0
##          C    0    8 1017    1    0
##          D    0    0   10  953    1
##          E    0    0    1    0 1081
##
## Overall Statistics
##
##                Accuracy : 0.9949
##                  95% CI : (0.9927, 0.9966)
##     No Information Rate : 0.2855
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9936
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9964   0.9930   0.9864   0.9990   0.9991
## Specificity            1.0000   0.9981   0.9981   0.9978   0.9998
## Pos Pred Value         1.0000   0.9921   0.9912   0.9886   0.9991
## Neg Pred Value         0.9986   0.9983   0.9971   0.9998   0.9998
## Prevalence             0.2855   0.1934   0.1752   0.1621   0.1839
## Detection Rate         0.2845   0.1920   0.1728   0.1619   0.1837
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9982   0.9955   0.9923   0.9984   0.9994
```

```
(accur_rf <- confM_rf$overall[1])
```

```
##  Accuracy
## 0.9949023
```

Random forest yielded much better results than classification trees. The accuracy here is 0.995, and so the out-of-sample error rate is 0.005.

Prediction on the testing set

We use random forests, which yielded much better in-sample results, to predict the outcome variable `classe` for the testing set.

```
(predict(fit_rf, testing))
```

```
##  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
```

End!