Machine Learning Project

Introduction

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit 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 be to use data from accelerometers on the `belt`, `forearm`, `arm`, and `dumbell` of `6 participant`. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. The five ways are exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E).

Only Class A corresponds to correct performance. The goal of this project is to predict the manner in which they did the exercise, i.e., Class A to E. 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).

Libraries

```r
rm(list = ls())
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

GetData

```r
# import the data from the URLs
trainurl <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testurl <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

training <- read.csv(trainurl,
                     na.strings = c("NA", "#DIV/0!", ""),
                     header = TRUE)
training$X <- NULL
write.csv(training, file = "../data/final_training.csv")

testing <- read.csv(testurl,
                     na.strings = c("NA", "#DIV/0!", ""),
                     header = TRUE)
```

```r
testing$X <- NULL
write.csv(testing, file = "../data/final_testing.csv")
```

Exploratory Analysis

Lets see what have in the data. The training dataset has 19622 observations and 159 variables, and the testing data set contains 20 observations and the same variables as the training set. We are trying to predict the outcome of the variable classe in the training set.

Import data

```r
# load data locally
training <- read.csv("../data/final_training.csv", na.strings = c("NA", ""))
training$X <- NULL
dim(training)
```

```
## [1] 19622   159
```

```r
testing <- read.csv("../data/final_testing.csv", na.strings = c("NA", ""))
testing$X <- NULL
dim(testing)
```

```
## [1]  20 159
```

Data Cleaning

We see there are many variables have more than 90% missing data. I hide the output since there are so many variables.

We should keep features that has no missing data, delete those having missing data in the training data. Then drop those variables in the test data.

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

Testing data and training data two different variables. Classe is only in training data, and problem_id is only in testing data. Classe is the target variable

```r
setdiff(names(training), names(testing))
```

```
## [1] "classe"
```

```r
setdiff(names(testing), names(training))
```

```
## [1] "problem_id"
```

Let's see what we have in classe variable.

```r
table(training$classe)
```

```
## 
##    A    B    C    D    E 
## 5580 3797 3422 3216 3607
```

There are some other variables that has low predicting power. These are `user_name`, timestamp variables, new_window, num_window. We can drop these variables.

```r
table(training$user_name)
```

```
##
##   adelmo carlitos  charles   eurico   jeremy    pedro
##     3892     3112     3536     3070     3402     2610
```

```
table(training$new_window)
```

```
##
##    no   yes
## 19216   406
```

```
training_df <- training[, -c(1:6)]
dim(training_df)
```

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

```
testing_df<- testing[, -c(1:6)]
dim(testing_df)
```

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

The cleaned data sets trainData and testData both have 53 columns with the same first 52 variables and the last variable classe and problem_id individually. trainData has 19622 rows while testData has 20 rows.

Prediction Algorithms

Data splitting

In order to get out-of-sample errors, we split the cleaned training set trainData into a training set (train, 70%) for prediction and a validation set (valid 30%) to compute the out-of-sample errors.

```
set.seed(7826)
inTrain <- createDataPartition(training_df$classe, p = 0.7, list = FALSE)
train <- training_df[inTrain, ]
valid <- training_df[-inTrain, ]
```

Algorithms

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

Classification trees

In practice, k=10 when doing k-fold cross validation. Here we consider 10-fold cross validation (default setting in trainControl function is 10) when implementing the algorithm.

Classification trees are **non-linear** models, data transformations, especially, monotone transformations may be less important (transformations that does not change the order of the values)

```

control <- trainControl(method = "cv", number = 10)
```

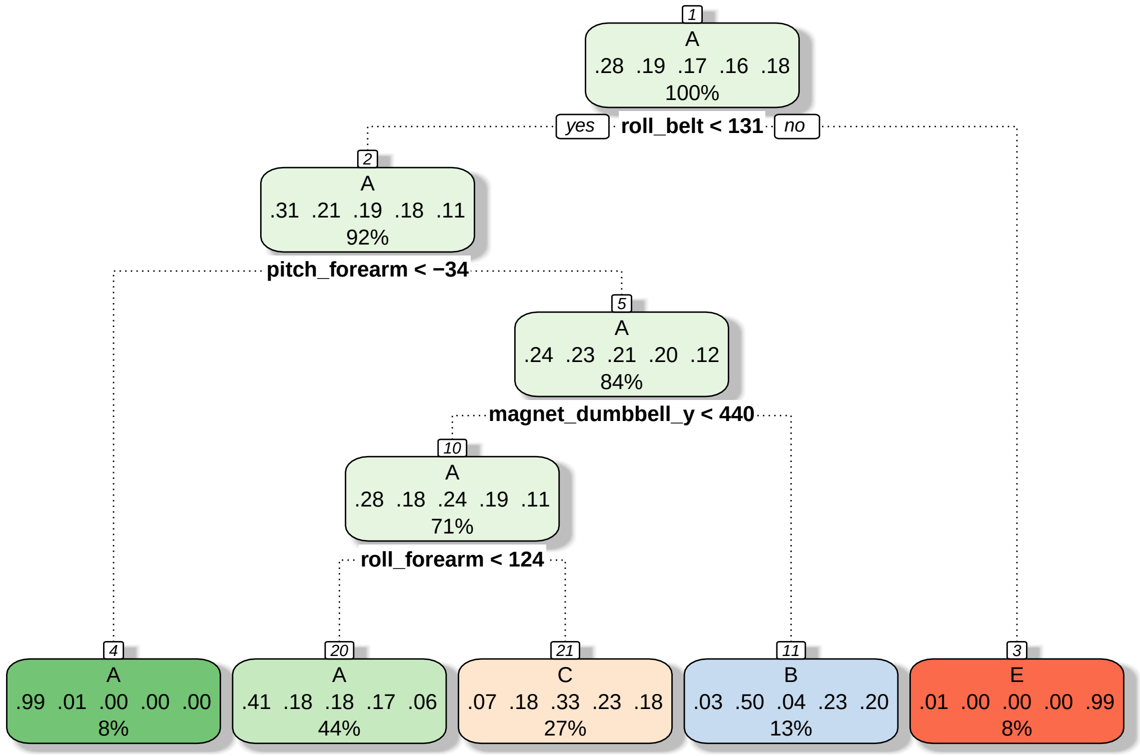
Cross-validation

```

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

print(fit_rpart, digits = 3)
```

Training the data

```
## CART
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 12363, 12365, 12363, 12364, 12362, 12366, ...
## Resampling results across tuning parameters:
##
##   cp      Accuracy  Kappa
##   0.0310  0.518     0.3696
##   0.0595  0.391     0.1661
##   0.1159  0.324     0.0597
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.031.
```

```
library(rpart)
library(rattle)

fancyRpartPlot(fit_rpart$finalModel)
```



Rattle 2022−Jan−07 21:59:19 d842a922

Plot the tree

```
# predict outcomes using validation set
predict_rpart <- predict(fit_rpart, newdata = valid)

# Show prediction result
conf_rpart <- confusionMatrix(data = predict_rpart,
                              reference =  factor(valid$classe))
conf_rpart
```

Validation

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1510  464  467  432  164
##          B   27  410   37  163  144
##          C  134  265  522  369  293
##          D    0    0    0    0    0
##          E    3    0    0    0  481
##
## Overall Statistics
##
##                Accuracy : 0.4967
##                  95% CI : (0.4838, 0.5095)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3425
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9020  0.35996   0.5088   0.0000  0.44455
## Specificity            0.6374  0.92183   0.7816   1.0000  0.99938
## Pos Pred Value         0.4972  0.52497   0.3298      NaN  0.99380
## Neg Pred Value         0.9424  0.85717   0.8828   0.8362  0.88872
## Prevalence             0.2845  0.19354   0.1743   0.1638  0.18386
## Detection Rate         0.2566  0.06967   0.0887   0.0000  0.08173
## Detection Prevalence   0.5161  0.13271   0.2690   0.0000  0.08224
## Balanced Accuracy      0.7697  0.64090   0.6452   0.5000  0.72196
```

```
(accuracy_rpart <- conf_rpart$overall[1])
```

Overall Accuracy

```
##  Accuracy
## 0.4966865
```

From the confusion matrix, the accuracy rate is 0.5, and so the out-of-sample error rate is 0.5. We can say that using classification tree does not predict the outcome `classe` very well.

Random forests

Since classification tree method does not perform well, we try random forest method instead.

```
fit_rf <- train(classe ~ .,
                data = train,
                method = "rf",
                trControl = control)  ## cross validation k=10
print(fit_rf, digits = 3)
```

Training the model

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 12364, 12363, 12364, 12362, 12362, 12363, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa
##    2    0.992     0.990
##   27    0.993     0.991
##   52    0.986     0.983
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.
```

```
# predict outcomes using validation set
predict_rf <- predict(fit_rf, valid)
```

```
# Show prediction result
(conf_rf <- confusionMatrix(data =  predict_rf,
                            reference = factor(valid$classe)))
```

Prediction on Valid data (Validation)

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1671    4    0    0    0
##          B    2 1134   11    0    1
##          C    0    1 1011   16    0
##          D    0    0    4  947    9
##          E    1    0    0    1 1072
##
## Overall Statistics
##
##                Accuracy : 0.9915
##                  95% CI : (0.9888, 0.9937)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9893
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9982   0.9956   0.9854   0.9824   0.9908
## Specificity            0.9991   0.9971   0.9965   0.9974   0.9996
## Pos Pred Value         0.9976   0.9878   0.9835   0.9865   0.9981
## Neg Pred Value         0.9993   0.9989   0.9969   0.9965   0.9979
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2839   0.1927   0.1718   0.1609   0.1822
## Detection Prevalence   0.2846   0.1951   0.1747   0.1631   0.1825
## Balanced Accuracy      0.9986   0.9963   0.9909   0.9899   0.9952
```

```
(accuracy_rf <- conf_rf$overall[1])
```

Overall Accuracy

```
##  Accuracy
## 0.9915038
```

Very high accuracy, indeed. Random forest algorithm predicts way better than classification tree algorithm.

The out-of-sample error rate is 0.008.

This may be due to the fact that many predictors are highly correlated. Random forests chooses a subset of predictors at each split and decorrelate the trees. This leads to high accuracy, although this algorithm is sometimes difficult to interpret and computationally inefficient.

Prediction on Testing Set

Let's use random forests to predict the outcome variable `classe` for the testing set. Since this data does not have `classe`, we can just predict. We can't validate the result.

```
(predict(fit_rf, testing_df))
```

```
##  [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
```