Coursera Practical Machine Learning Final Project

Anirban Chatterjee

11/11/2020

Overview

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. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways.

The data consists of a Training data and a Test data (to be used to validate the selected model).

The goal of your project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set. You may use any of the other variables to predict with.

Data Loading and Processing

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.0.3
```

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

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

```
library(rpart)
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.0.3
```

```
library(RColorBrewer)
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 4.0.3
```

```
## Loading required package: tibble
```

```
## Loading required package: bitops

## Rattle: A free graphical interface for data science with R.
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.0.3

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:rattle':
##
##     importance

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 4.0.3

## corrplot 0.84 loaded
```

```
library(repmis)
```

```
## Warning: package 'repmis' was built under R version 4.0.3
```

Getting, Cleaning and Exploring the data

```
# 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 <- source_data(trainurl, na.strings = c("NA", "#DIV/0!", ""), header = TRUE)
# testing <- source_data(testurl, na.strings = c("NA", "#DIV/0!", ""), header = TRUE)
# load data locally
train_in <- read.csv("pml-training.csv", na.strings = c("NA", ""))
valid_in <- read.csv("pml-testing.csv", na.strings = c("NA", ""))
dim(train_in)
```

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

```
dim(valid_in)
```

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

As shown below there are 19622 observations and 160 variables in the Training dataset

Cleaning the input data

We remove the variables that contains missing values. Note along the cleaning process we display the dimension of the reduced dataset

```
trainData<- train_in[, colSums(is.na(train_in)) == 0]
validData <- valid_in[, colSums(is.na(valid_in)) == 0]
dim(trainData)
```

```
## [1] 19622    60
```

```
dim(validData)
```

```
## [1] 20 60
```

We now remove the first seven variables as they have little impact on the outcome classe

```
trainData <- trainData[, -c(1:7)]
validData <- validData[, -c(1:7)]
dim(trainData)
```

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

```
dim(validData)
```

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

Preparing the datasets for prediction

Preparing the data for prediction by splitting the training data into 70% as train data and 30% as test data. This splitting will server also to compute the out-of-sample errors.

The test data renamed: valid_in (validate data) will stay as is and will be used later to test the prodction algorithm on the 20 cases

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

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

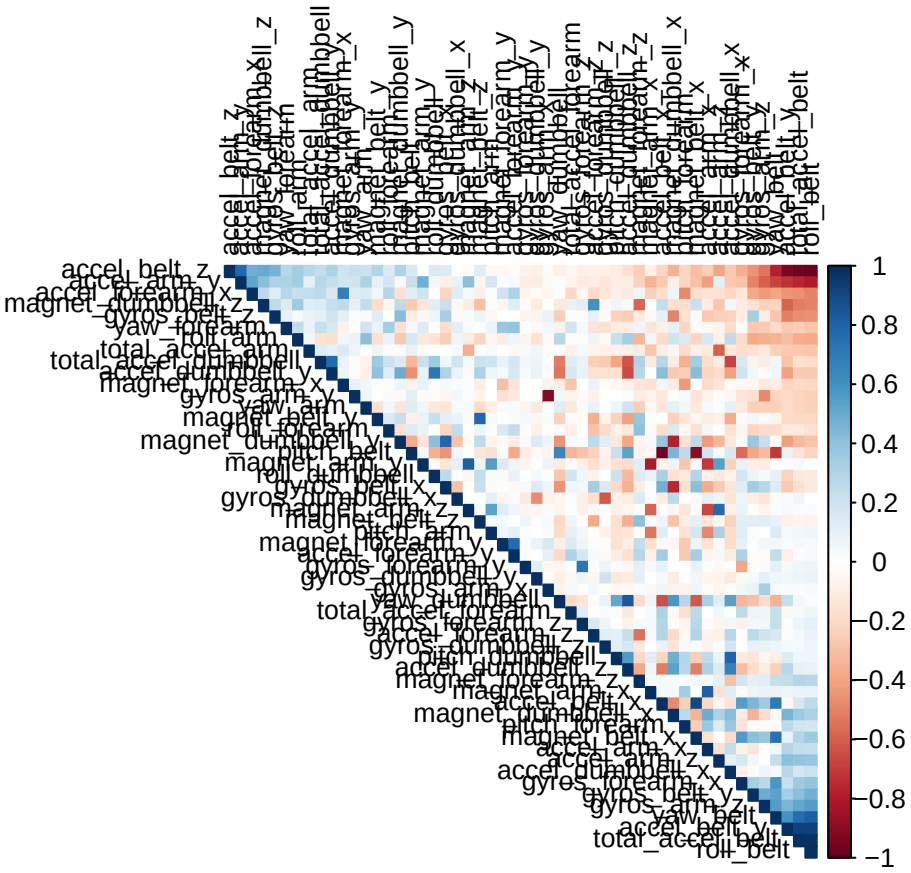
```
dim(valid)
```

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

After this cleaning we are down now to 53 variables

The following correlation plot uses the following parameters (source:CRAN Package 'corrplot') "FPC": the first principal component order. "AOE": the angular order tl.cex Numeric, for the size of text label (variable names) tl.col The color of text label.

```
cor_mat <- cor(train[, -53])
corrplot(cor_mat, order = "FPC", method = "color", type = "upper",
         tl.cex = 0.8, tl.col = rgb(0, 0, 0))
```



Prediction Algorithm

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

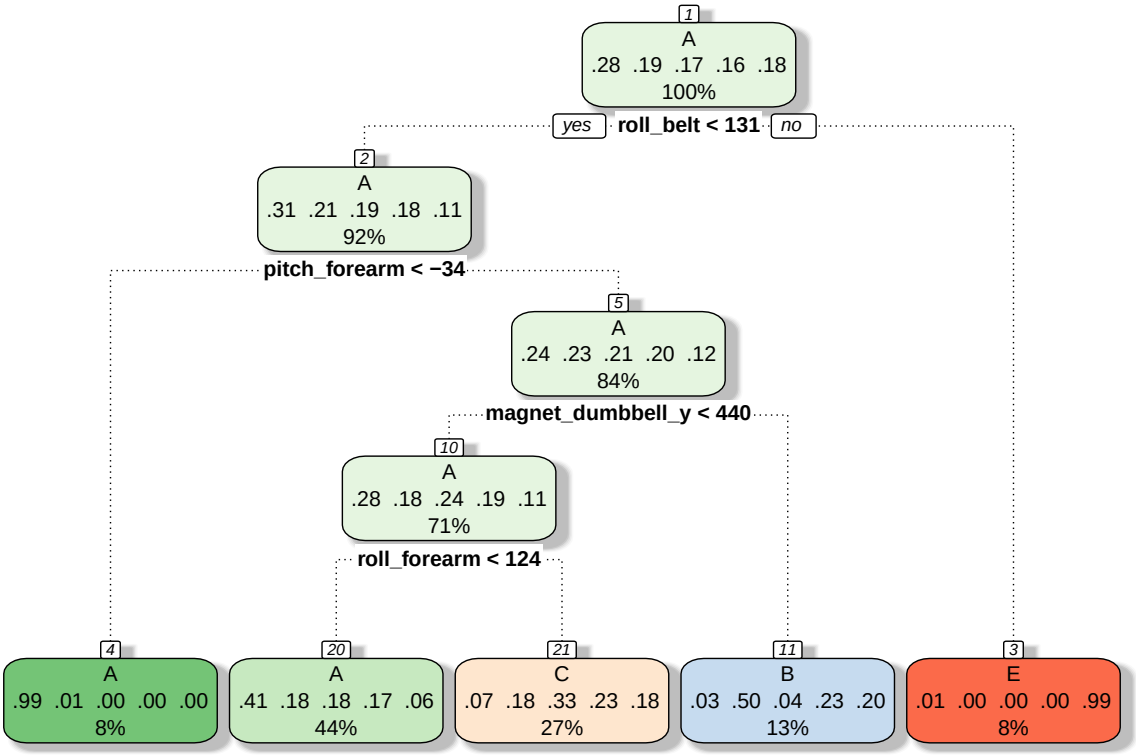
Classification Trees

In practice, k=5 or k=10 when doing k-fold cross validation. Here we consider 5-fold cross validation (default setting in trainControl function is 10) when implementing the algorithm to save a little computing time. Since data transformations may be less important in non-linear models like classification trees, we do not transform any variables.

```
control <- trainControl(method = "cv", number = 5)
fit_rpart <- train(classe ~ ., data = train, method = "rpart",
                   trControl = control)
print(fit_rpart, digits = 4)
```

```
## CART
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10990, 10990, 10990, 10989, 10989
## Resampling results across tuning parameters:
##
##   cp       Accuracy  Kappa
##   0.03102  0.5260    0.38038
##   0.05954  0.3935    0.16982
##   0.11586  0.3168    0.04946
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.03102.
```

```
fancyRpartPlot(fit_rpart$finalModel)
```



Rattle 2020−Nov−12 14:55:06 Anirban Chatterjee

```r
# predict outcomes using validation set
predict_rpart <- predict(fit_rpart, valid)
# Show prediction result
conf_rpart <- confusionMatrix(predict_rpart, factor(valid$classe))
conf_rpart
```

```
## 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
```

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

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

From the confusion matrix, the accuracy rate is 0.5, and so the out-of-sample error rate is 0.5. Using classification tree does not predict the outcome classe very well.

Random Forest

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

```r
fit_rf <- train(classe ~ ., data = train, method = "rf", trControl = control)
print(fit_rf, digits = 4)
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10991, 10990, 10990, 10988, 10989
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa
##    2    0.9913    0.9889
##   27    0.9921    0.9900
##   52    0.9840    0.9797
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.
```

```r
# predict outcomes using validation set
predict_rf <- predict(fit_rf, valid)
# Show prediction result
conf_rf <- confusionMatrix(predict_rf, factor(valid$classe))
conf_rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1671    4    0    0    0
##          B    2 1134    8    0    1
##          C    0    1 1013   16    1
##          D    0    0    5  947    9
##          E    1    0    0    1 1071
##
## Overall Statistics
##
##                Accuracy : 0.9917
##                  95% CI : (0.989, 0.9938)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9895
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9982   0.9956   0.9873   0.9824   0.9898
```

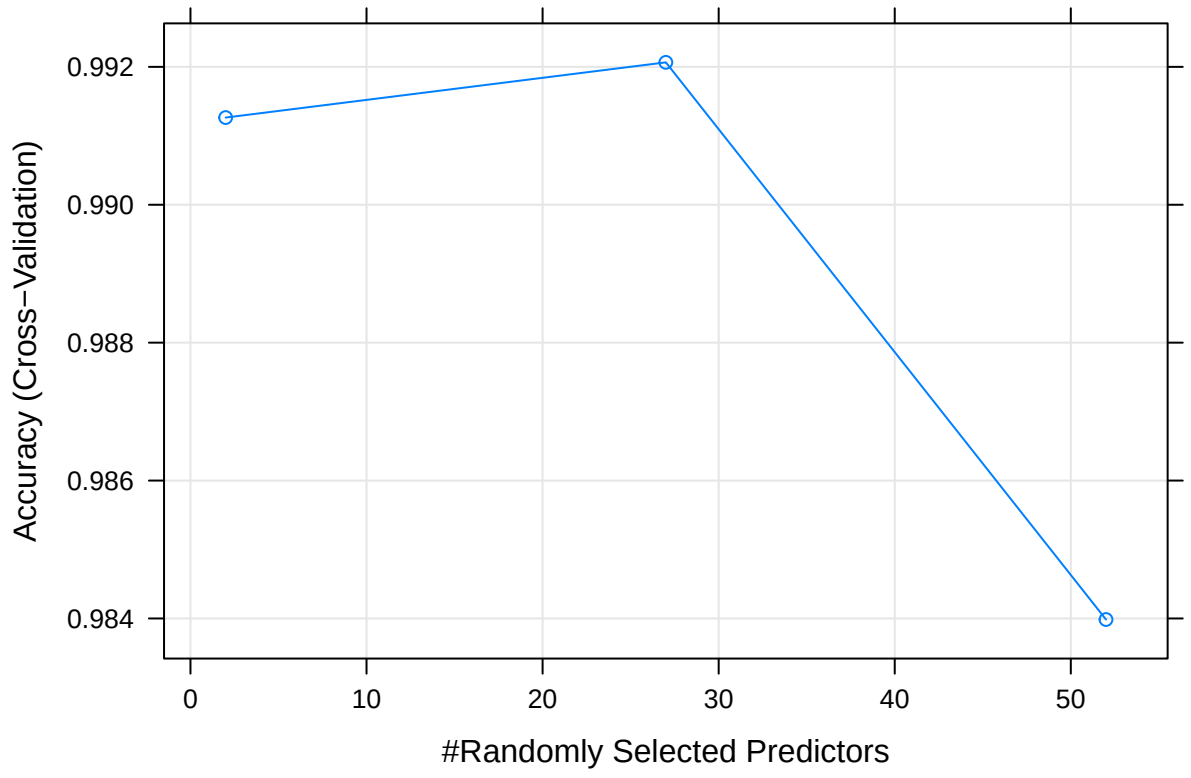
```
## Specificity            0.9991   0.9977   0.9963   0.9972   0.9996
## Pos Pred Value         0.9976   0.9904   0.9825   0.9854   0.9981
## Neg Pred Value         0.9993   0.9989   0.9973   0.9965   0.9977
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2839   0.1927   0.1721   0.1609   0.1820
## Detection Prevalence   0.2846   0.1946   0.1752   0.1633   0.1823
## Balanced Accuracy      0.9986   0.9966   0.9918   0.9898   0.9947
```

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

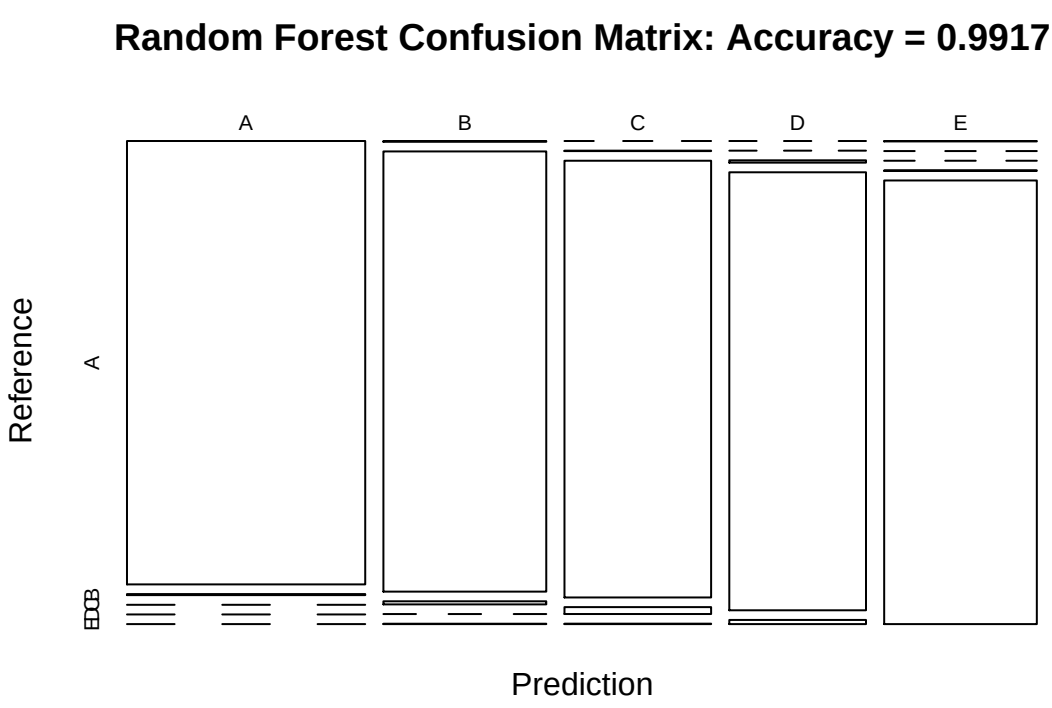
```
##  Accuracy
## 0.9916737
```

For this dataset, random forest method is way better than classification tree method. The accuracy rate is 0.992, and so 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.

```r
plot(fit_rf)
```




```r
plot(conf_rf$table, col = conf_rf$byClass, main = paste("Random Forest Confusion Matrix: Accuracy =", r
```



The accuracy rate using the random forest is very high

Applying the best model to the validation data

By comparing the accuracy rate values of the two models, it is clear the the 'Random Forest' model is the winner. So will use it on the validation data

```
Results <- predict(fit_rf, newdata=validData)
Results
```

```
##  [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 Results output will be used to answer the "Course Project Prediction Quiz"