Practical Machine Learning: Final-Project

August 3rd, 2019

Introduction

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.

Project Background and Executive Summary

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. Follwoing are the steps we take to lead the final results.

Data processing

```
# Loading required r-packages:

library(caret); library(rattle); library(rpart); library(rpart.plot)
```

```
## Loading required package: lattice

## Loading required package: ggplot2

## Rattle: A free graphical interface for data science with R.
## Version 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(randomForest); library(repmis)
```

```
## 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
```

```
training <- read.csv("pml-training.csv", na.strings = c("NA", ""))
testing <- read.csv("pml-testing.csv", na.strings = c("NA", ""))
```

Look at the dimensions & head of the dataset to get an idea:

```r
# dimension of training data
dim(training)
```

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

```r
# dimension of testing data
dim(testing)
```

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

```r
# Excluded because excessive amount of data
# head(training)
# Excluded because excessivness
#str(training,20)
# Excluded because excessivness
#summary(training)
```

As the dimension of trating and test sets show: The training dataset has 19622 observations and 160 variables, and the testing data set contains 20 observations and the same variables as the training set. We are trying to predict the outcome variable named "classe"" in the training set.

Data Cleaning:

We now delete columns (predictors) of the training set that contain any missing values.

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

As the data shows, we can remove the first seven predictors since these variables have little predicting power for the outcome classe.

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

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

```r
dim(testData)
```

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

#Preparing Data to perform Machine Learning Algorithms In order to get out-of-sample errors we need to have a validation set(test set into training set). Therefore, 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.

Data Splitting

```r
set.seed(1226)
inTrain <- createDataPartition(trainData$classe, p = 0.7, list = FALSE)
train <- trainData[inTrain, ]
valid <- trainData[-inTrain, ]
```

Applying Prediction Algorithms

In this section we apply both classification trees and random forest to see the results and do the comparisions.

First Approach: Classification Trees

Here we apply k-fold algorithm for cross-validation. We apply K=5 (5-fold) cross-validation.

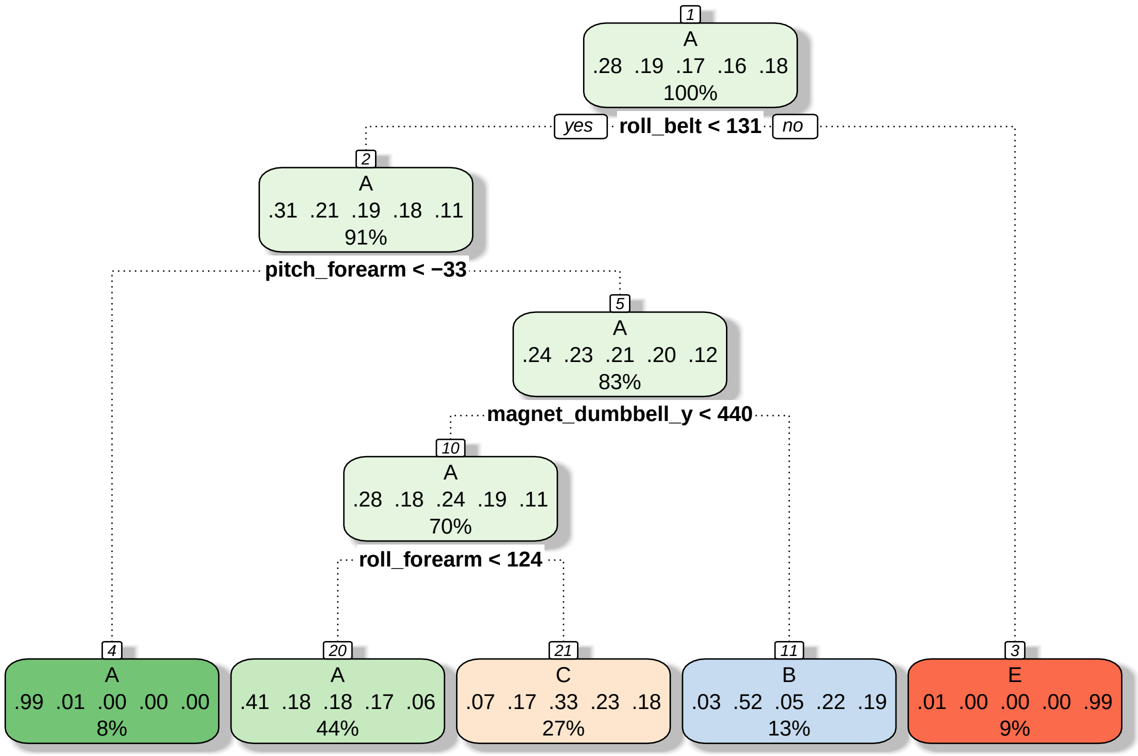
```
# Set 5-fold cross valication
control <- trainControl(method = "cv", number = 5)

# Applying classification trees on train data set
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: 10989, 10988, 10990, 10990, 10991
## Resampling results across tuning parameters:
##
##   cp       Accuracy  Kappa
##   0.03438  0.5105    0.36060
##   0.06123  0.4461    0.25774
##   0.11850  0.3169    0.04956
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.03438.
```

See how the classification tree looks like in training set:

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



Rattle 2019−Aug−03 14:30:53 Arezou

See how the prediction works on validation data set. and compute the accuracy:

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1519   28  125    0    2
##          B  489  374  276    0    0
##          C  492   28  506    0    0
##          D  441  185  338    0    0
##          E  185  152  291    0  454
##
## Overall Statistics
##
##                Accuracy : 0.4848
##                  95% CI : (0.4719, 0.4977)
##     No Information Rate : 0.5312
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3256
##
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.4859  0.48761  0.32943       NA  0.99561
## Specificity            0.9438  0.85053  0.88043   0.8362  0.88432
## Pos Pred Value         0.9074  0.32836  0.49318       NA  0.41959
## Neg Pred Value         0.6184  0.91719  0.78802       NA  0.99958
## Prevalence             0.5312  0.13033  0.26100   0.0000  0.07749
## Detection Rate         0.2581  0.06355  0.08598   0.0000  0.07715
## Detection Prevalence   0.2845  0.19354  0.17434   0.1638  0.18386
## Balanced Accuracy      0.7149  0.66907  0.60493       NA  0.93997
```

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

```
##  Accuracy
## 0.4847918
```

Result:

The results of confusion matrix shows the accuracy rate is 0.5, and so the out-of-sample error rate is 0.5. Therefore, we can see that using classification tree does not predict the outcome classe very well.

Second Approach: Random Forest

```
fit_rf <- train(classe ~ ., data = train, method = "rf",
                   trControl = control, tuneLength=1)
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: 10990, 10990, 10990, 10990, 10988
## Resampling results:
##
##   Accuracy  Kappa
##   0.9937    0.9921
##
## Tuning parameter 'mtry' was held constant at a value of 17
```

See how the prediction works on validation data set. and compute the accuracy:

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

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    0    0    0    1
##          B   12 1124    3    0    0
##          C    0    3 1015    8    0
##          D    0    0   10  954    0
##          E    0    0    5    3 1074
##
## Overall Statistics
##
##                Accuracy : 0.9924
##                  95% CI : (0.9898, 0.9944)
##     No Information Rate : 0.2863
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9903
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9929   0.9973   0.9826   0.9886   0.9991
## Specificity            0.9998   0.9968   0.9977   0.9980   0.9983
## Pos Pred Value         0.9994   0.9868   0.9893   0.9896   0.9926
## Neg Pred Value         0.9972   0.9994   0.9963   0.9978   0.9998
## Prevalence             0.2863   0.1915   0.1755   0.1640   0.1827
## Detection Rate         0.2843   0.1910   0.1725   0.1621   0.1825
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9963   0.9971   0.9902   0.9933   0.9987
```

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

```
##  Accuracy
## 0.9923534
```

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

Prediction on Testing Set

We now use random forests to predict the outcome variable classe for the testing set.

```
(predict(fit_rf, testData))
```

```
##  [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
```

Comparison of prediction results using both classification algorithms:

It is nice to see the comparison of predictions of both classification algorithms (classification trees and random forest) on the testing set:

```r
predictions <- t(cbind(
    randon_forest=as.data.frame(predict(fit_rf, testData), optional=TRUE),
    classification_trees=as.data.frame(predict(fit_rpart, testData), optional=TRUE)))
predictions
```

```
##                      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## randon_forest        "B"  "A"  "B"  "A"  "A"  "E"  "D"  "B"  "A"  "A"
## classification_trees "C"  "A"  "C"  "A"  "A"  "C"  "C"  "A"  "A"  "A"
##                      [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19]
## randon_forest        "B"   "C"   "B"   "A"   "E"   "E"   "A"   "B"   "B"
## classification_trees "C"   "C"   "C"   "A"   "C"   "A"   "A"   "A"   "A"
##                      [,20]
## randon_forest        "B"
## classification_trees "C"
```