Practical Maching Learning Project

Bikram Bhusal

November 11, 2019

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 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

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.

Getting Data

```
trainD<-read.csv ("pml-training.csv",header = TRUE)
validD<-read.csv ("pml-testing.csv",header = TRUE)
dim(trainD)
```

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

```
dim(validD)
```

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

some useful packages:

```
library(caret)
```

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

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

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

Cleaning and Preparing the Data:

Note along the cleaning process we display the dimension of the reduced dataset Here,

```
# Removing the variables that contains missing values.
trainD<- trainD[, colSums(is.na(trainD)) == 0]
validD <- validD[, colSums(is.na(validD)) == 0]
dim(trainD)

## [1] 19622    93

dim(validD)

## [1] 20 60
```

and

```
# removing the first seven variables as they have little impact on the
outcome classe
trainD <- trainD[, -c(1:7)]
validD <- validD[, -c(1:7)]
dim(trainD)

## [1] 19622    86

dim(validD)

## [1] 20 53
```

Preparing the datasets for prediction:

```
set.seed(123)
R_training <- createDataPartition(y=trainD$classe, p=0.7, list=F)
training <- trainD[R_training, ]
testing <- trainD[-R_training, ]
```

Further, cleaning the data by removing the variables that are nearly zero variance

```
# remove variables with nearly zero variance
nzv <- nearZeroVar(training)
training <- training[, -nzv]
testing <- testing[, -nzv]
dim(training)

## [1] 13737    53

dim(testing)

## [1] 5885   53
```

After this cleaning we are down now to 53 variables

Model Building

Begin with buildilding a model using classification trees.

Here is a classification tree:

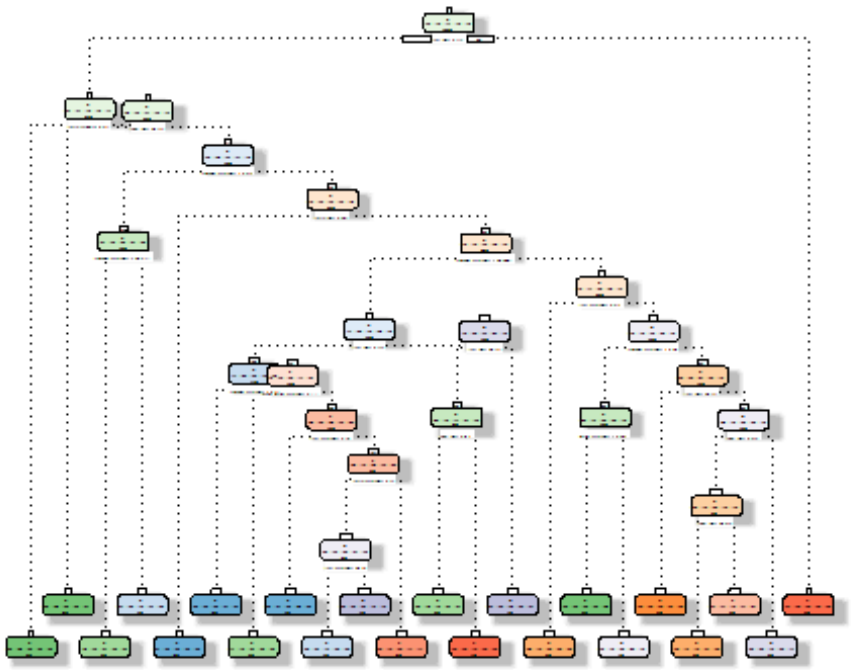
```
library(rattle)

## 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(rpart)

set.seed(345)
modelfit_tree <- rpart(classe ~ ., data=training, method="class")
fancyRpartPlot(modelfit_tree)

## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



We then validate the model "modelfit_tree" on the testData to find out how well it performs by looking at the accuracy variable:

```
predict_tree <- predict(modelfit_tree, testing, type = "class")
cm_tree <- confusionMatrix(predict_tree, testing$classe)
cm_tree

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1424  184   52   56   25
##          B   26  667   54   48   70
##          C   33  107  701   53   52
```

```
##          D  167  106  193  735  157
##          E   24   75   26   72  778
##
## Overall Statistics
##
##                Accuracy : 0.7315
##                  95% CI : (0.72, 0.7428)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6606
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8507   0.5856   0.6832   0.7624   0.7190
## Specificity            0.9247   0.9583   0.9496   0.8734   0.9590
## Pos Pred Value         0.8179   0.7711   0.7410   0.5412   0.7979
## Neg Pred Value         0.9397   0.9060   0.9342   0.9494   0.9381
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2420   0.1133   0.1191   0.1249   0.1322
## Detection Prevalence   0.2958   0.1470   0.1607   0.2308   0.1657
## Balanced Accuracy      0.8877   0.7719   0.8164   0.8179   0.8390
```

We see that the accuracy rate of the model is low: 0.7315 and therefore the out-of-sample-error is about 0.27 which is considerable.

Random Forests model

```
library(randomForest)
```

```
## 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
```

Fitting the model

```
controlRF <- trainControl(method="cv", number=3, verboseIter=FALSE)
modRF1 <- train(classe ~ ., data=training, method="rf", trControl=controlRF)
modRF1$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 0.67%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3901    3    1    0    1 0.001280082
## B   23 2629    5    1    0 0.010910459
## C    0   12 2379    5    0 0.007095159
## D    0    0   27 2224    1 0.012433393
## E    0    1    5    7 2512 0.005148515
```

I see that it decided to use 500 trees and try 27 variables at each split.

Model Evaluation:

```
predictRF1 <- predict(modRF1, newdata=testing)
cmrf <- confusionMatrix(predictRF1, testing$classe)
cmrf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1671    8    0    0    0
##          B    3 1130    8    0    0
##          C    0    1 1015   19    2
##          D    0    0    3  944    1
##          E    0    0    0    1 1079
##
## Overall Statistics
##
##                Accuracy : 0.9922
##                  95% CI : (0.9896, 0.9943)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9901
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
```

```
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9982   0.9921   0.9893   0.9793   0.9972
## Specificity            0.9981   0.9977   0.9955   0.9992   0.9998
## Pos Pred Value         0.9952   0.9904   0.9788   0.9958   0.9991
## Neg Pred Value         0.9993   0.9981   0.9977   0.9959   0.9994
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2839   0.1920   0.1725   0.1604   0.1833
## Detection Prevalence   0.2853   0.1939   0.1762   0.1611   0.1835
## Balanced Accuracy      0.9982   0.9949   0.9924   0.9892   0.9985
```

The accuracy rate using the random forest is very high: Accuracy : 0.9922 and therefore the out-of-sample-error is equal to 0.0078.This is an excellent result, so rather than trying additional algorithms, I will use Random Forests to predict on the test set.

Predictions

All that is left is to use this model to predict the classes of the validation data:

```
pred_val <- predict(modRF1, validD)
pred_val
```

```
##  [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
```