Practical Machine Learning Project

Nicholas Trankle

02/07/2020

This document contains the work as required by the Practical Machine Learning course on Coursera.

The project background brief was given as follows:

"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, your goal will be 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."

Project Brief

The **aim** of the project is to use training data to build a model that will classify the manner in which the exercise was performed. Using R's **caret** package, we will build two models, a decision tree model and a random forest model. Since the size of the data is so large, the training data will be partitioned into a 60% model training set and a 40% model testing set. This will suffice as model cross validation. We will then report on the expected out of sample error. The more accurate model will then be chosen and subjected to the test data provided to classify the manner in which the exercises were performed.

The project will be divided into a number of steps which will be summarised as follows: 1.) Cleaning the data. 2.) Dividing the training data into a training and testing set. 3.) Create a model using the decision tree algorithm and test it's capabilities. 4.) Create another model using the random forest algorithm and test it's capabilities. 5.) Use the better of the two models to make predictions on the test data.

```
library("caret")
```

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

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

```
library("rpart")
library("rpart.plot")
library("RColorBrewer")
library("rattle")
```

```
## 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")
```

```
## 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
```

```
ProjTrain = read.csv("C:/Users/gct1/Desktop/pml-training.csv")
ProjTest = read.csv("C:/Users/gct1/Desktop/pml-testing.csv")
```

The data have been downloaded and stored into dataframes ProjTrain and ProjTest.

1.) Cleaning the data

After having analysed the raw data, there are a three pertinent observations that can be made.

1.) The first column ("X") seems to be an ID column. In other words, it simply acts as a reference to a specific entry. Although this may be useful in some instances, it should not be included in the data sent to the model as it has no influence on the data and the manner (classe) in which the movement was done.

```
ProjTrain <- ProjTrain[c(-1)]
dim(ProjTrain) # checking to see that a (the ID) column has been removed.
```

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

2.) The data are analysed to check if there are any variables that have near zero variance. R's function NearZeroVariance() provides a useful summary, indicating which variables could be excluded from that model. These variables are then excluded from the dataframe which will be fed into the model.

```
NZV <- nearZeroVar(ProjTrain, saveMetrics = T)
NZV <- NZV[NZV$nzv == 1,]
NZV_RowNames <- row.names(NZV)
ProjTrain <- subset(ProjTrain, select = !names(ProjTrain) %in% NZV_RowNames)
dim(ProjTrain) # checking the dimensions of ProjTrain
```

```
## [1] 19622    99
```

3.) Many of the variables contain a lot of NA entries. These variables could also render the model to be inaccurate. As such, a decision is made to exclude variables which contain more than 60% NA's.

```
temp <- ProjTrain  # a necessary step as we will remove columns which makes loop indexing complicated
for(i in 1:length(ProjTrain)){   # loop for each column
  if(sum(is.na(ProjTrain[,i]))/nrow(ProjTrain) >= 0.6){   # evaluates if more than 60% NA's
    for(j in 1:length(temp)){
      if(length(grep(names(ProjTrain[i]), names(temp)[j])) == 1){   # removes the column off of temp
        temp <- temp[, -j]
      }
    }
  }
}
ProjTrain <- temp   # updating ProjTrain after the applicable variables have been removed.
dim(ProjTrain)   # Checking ProjTrain to see how many variables are left
```

```
## [1] 19622    58
```

Since the training data has been cleaned for the model, we can also remove these columns from the test data. The remaining column names in modTraining will be extracted and applied to ProjTest.

```
modTraining_colNames <- colnames(ProjTrain)
# Since ProjTest does not include the column classe (it is replaced with the column problem_id), we wil
modTraining_colNames <- colnames(ProjTrain[,-58])
TestData <- ProjTest[modTraining_colNames]
dim(TestData)   # Check that TestData has been correctly modified
```

```
## [1] 20 57
```

2.) Divide training data into training and testing sets

As per standard best practices, the provided training data is partitioned into model training and model testing sets with a 60:40 ratio.

```
inTrain <- createDataPartition(y = ProjTrain$classe, p=0.6, list=FALSE)
modTraining <- ProjTrain[inTrain,]
modTesting <- ProjTrain[-inTrain,]
dim(modTraining)
```

```
## [1] 11776    58
```

```
dim(modTesting)
```

```
## [1] 7846   58
```

To ensure that there are no issues with the training versus testing data, the test data is coerced to be the same as the training data.

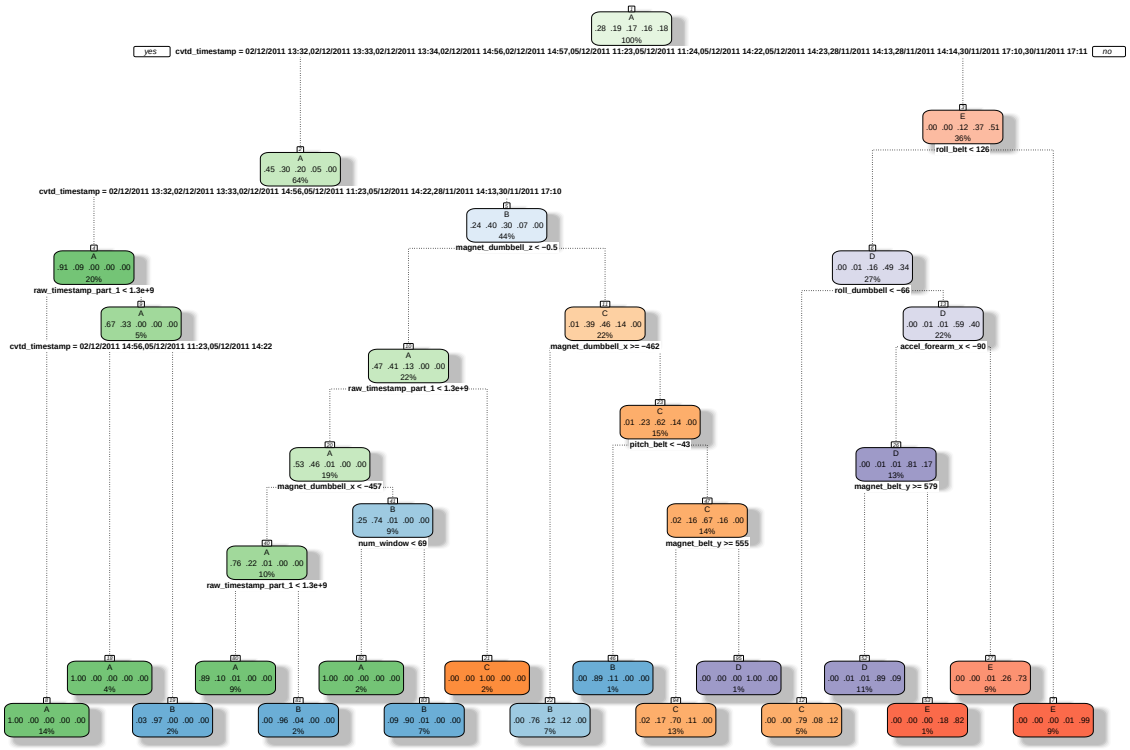
```
for(i in 1: length(TestData)){
  for(j in 1:length(modTraining)){
    if(length(grep(names(modTraining[i]), names(TestData)[j])) == 1){
      class(TestData[j]) <- class(modTraining[i])
    }
  }
}
```

3.) Create a model using the decision tree algorithm and test it's capabilities

```
dtModel <- rpart(classe ~ ., data = modTraining, method = "class")
```

A plot of the Decision Tree is given below:

```
fancyRpartPlot(dtModel)
```



The Decision Tree model is tested on the partitioned data modTesting and the results are given.

```
dtPredict <- predict(dtModel, modTesting, type = "class")
```

```
confusionMatrix(table(dtPredict, modTesting$classe))
```

```
## Confusion Matrix and Statistics
##
##
## dtPredict    A    B    C    D    E
##         A 2139   53    8    1    0
##         B   69 1275   80   57    0
##         C   24  180 1256  132   66
##         D    0   10   12  870   65
##         E    0    0   12  226 1311
##
## Overall Statistics
##
##                Accuracy : 0.8732
##                  95% CI : (0.8656, 0.8805)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8396
##
##  Mcnemar's Test P-Value : NA
##
```

```
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9583   0.8399   0.9181   0.6765   0.9092
## Specificity            0.9890   0.9674   0.9379   0.9867   0.9628
## Pos Pred Value         0.9718   0.8609   0.7575   0.9091   0.8464
## Neg Pred Value         0.9835   0.9618   0.9819   0.9396   0.9792
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2726   0.1625   0.1601   0.1109   0.1671
## Detection Prevalence   0.2805   0.1888   0.2113   0.1220   0.1974
## Balanced Accuracy      0.9736   0.9037   0.9280   0.8316   0.9360
```

As can be seen by the results, the out of sample accuracy of the decision tree model is 89.23%. The 95% confidence interval of this accuracy is 88.52% to 89.91%.

4.) Create another model using the random forest algorithm and test it's capabilities.

```
rfModel <- randomForest(as.factor(classe) ~ ., data = modTraining)
rfPredict <- predict(rfModel, modTesting, type = "class")
confusionMatrix(table(rfPredict, modTesting$classe))
```

```
## Confusion Matrix and Statistics
##
##
## rfPredict    A    B    C    D    E
##         A 2232    0    0    0    0
##         B    0 1518    2    0    0
##         C    0    0 1365    5    0
##         D    0    0    1 1281    0
##         E    0    0    0    0 1442
##
## Overall Statistics
##
##                Accuracy : 0.999
##                  95% CI : (0.998, 0.9996)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9987
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   0.9978   0.9961   1.0000
## Specificity            1.0000   0.9997   0.9992   0.9998   1.0000
## Pos Pred Value         1.0000   0.9987   0.9964   0.9992   1.0000
## Neg Pred Value         1.0000   1.0000   0.9995   0.9992   1.0000
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1935   0.1740   0.1633   0.1838
## Detection Prevalence   0.2845   0.1937   0.1746   0.1634   0.1838
## Balanced Accuracy      1.0000   0.9998   0.9985   0.9980   1.0000
```

The results of the random forest model indicate an out of sample accuracy of 99.87%, with a 95% confidence interval of 99.77 to 99.94%. Since this model is more accurate than the decision tree model, it is selected as the model of choice to evaluate the TestData.

5.) Use the better of the two models to make predictions on the test data.

The random forest model is used to predict the class of the provided test data.

```
TestPredictions <- predict(rfModel,ProjTest, type = "class")
TestPredictions
```

```
##  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
```