Practical Machine Learning Course Project

Daniel Arturo Lopez Sanchez

9/17/2020

Overview

This is the Course Project of the Practical Machine Learning course, as part of the Data Science Specialization by John Hopkins University. The purpose of the project is to train a model and predict the outcome on a validation test to answer the prediction quiz.

Backgound

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. More information is available from the website here: http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Preparing Enviroment

Loading the libraries we are going to need for the models

```r
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.0.2

## Loading required package: lattice

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.0.2
```

```r
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 4.0.2

## corrplot 0.84 loaded
```

```
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 4.0.2

## Loading required package: tibble

## Warning: package 'tibble' was built under R version 4.0.2

## 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(rpart)
```

Loading the training and test set given to us. We are going to change the name of the testing set to validation, and the training set to buildData.

```
buildData <- read.csv("pml-training.csv")
validation <- read.csv("pml-testing.csv")
```

Data slicing

Creating a Data Partition object for training and testing.

```
inTrain <- createDataPartition(y = buildData$classe, p = 0.7, list = F)
training <- buildData[inTrain, ]; testing <- buildData[-inTrain, ]
dim(training);dim(testing)
```

```
## [1] 13737   160

## [1] 5885  160
```

We can see that our training and testing sets have a lot of variables and not all of them are useful. We have to reduce the dimesionality to fit a good model.

Dimensionality Reduction

The techniques we are going to use for cleaning and reducing the number of predictor are going to be: Missing values ratio (threshold: 90% NA's on columns), Near Zero Variance, PCA - if needed.

Missing values

There are a lot of columns that contain NA's. With this algorithm we calculate the missing values ratio, and get rid of every column with a ratio higher than 0.9.

```r
NA_Threshold <- function(x){ mean(is.na(x)) }
ObjNA <- sapply(X = training, FUN = NA_Threshold) > 0.9
training<- training[, ObjNA==F]
testing<- testing[, ObjNA==F]
dim(training);dim(testing)
```

```
## [1] 13737    93
```

```
## [1] 5885   93
```

Near Zero Variance

To remove the empty spaces we are going to use the near zero variance technique. for this we'll use the fuction nearZeroVar()

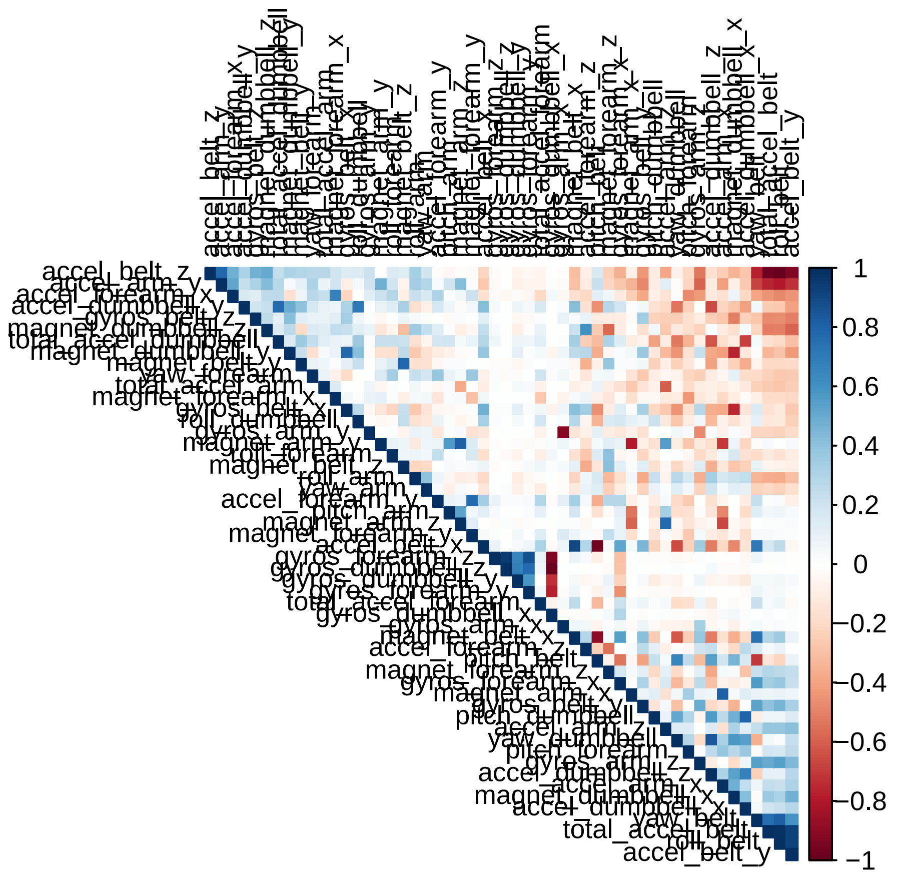
```r
nzv <- nearZeroVar(training)
training <- training[,-nzv]
testing <- testing[,-nzv]
```

Removing the first 6 columns, which aren't valuable for the analysis.

```r
training<- training[,-c(1:6)]
testing <- testing[,-c(1:6)]
```

Correlation Analysis

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



In the correlation matrix we see a few variables that seem highly correlated. We are going to create a pre-proccesing object with the method="pca" to perform a Principal Component Analysis. We'll evaluate our model with this analysis if we get a high accuracy; if not, we'll use the reduced sets before the Principal Components Analysis.

```
preProc <- preProcess(x = training[,-53], method = "pca")
trainingPC <- predict(preProc, training)
testingPC <- predict(preProc, testing)
```

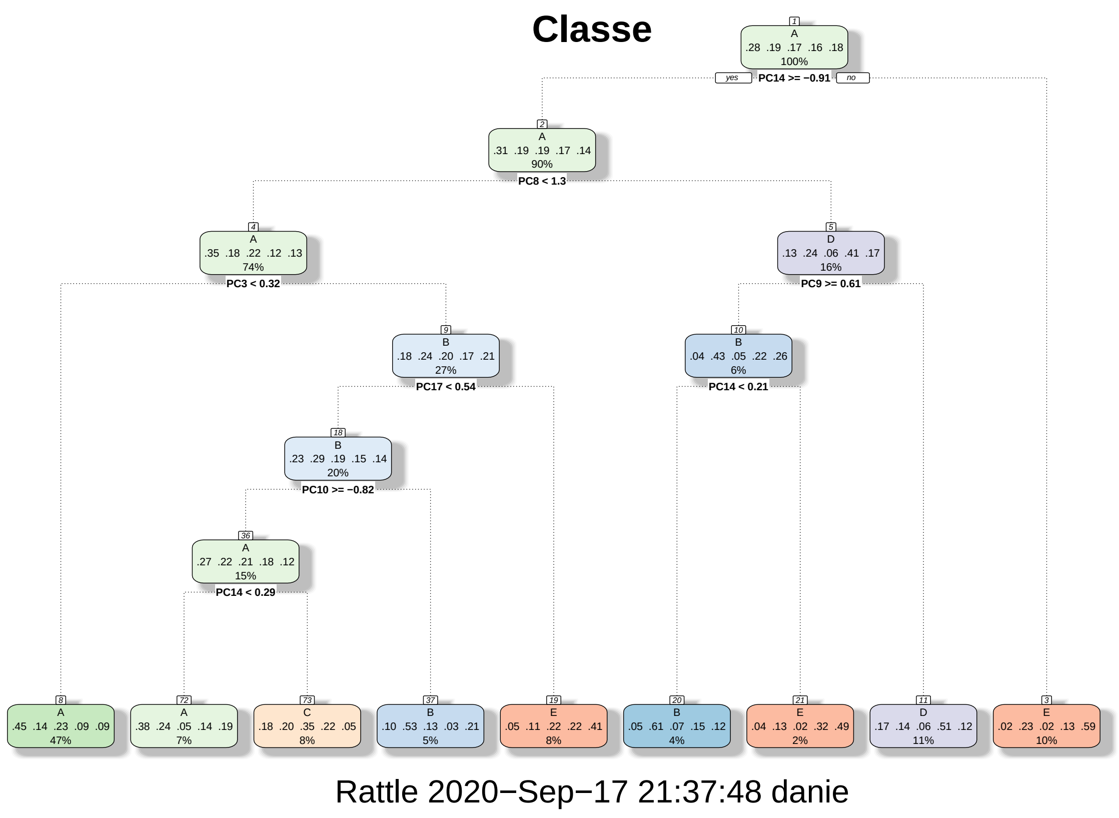
We are done with dimensionality reduction now, and ready to begin with our prediction model.

Prediction Model

In this part we are going to evaluate the best model to apply to the validation set. For this we are going to train models with different methods and estimate the out-of-sample error and accuracy. The methods we are going to use are: Decision Trees, Gradient Boosting and Random Forest.

Desicion Trees

```
modFit_DT <- rpart(classe~., data = trainingPC, method = "class")
fancyRpartPlot(modFit_DT, main = "Classe")
```



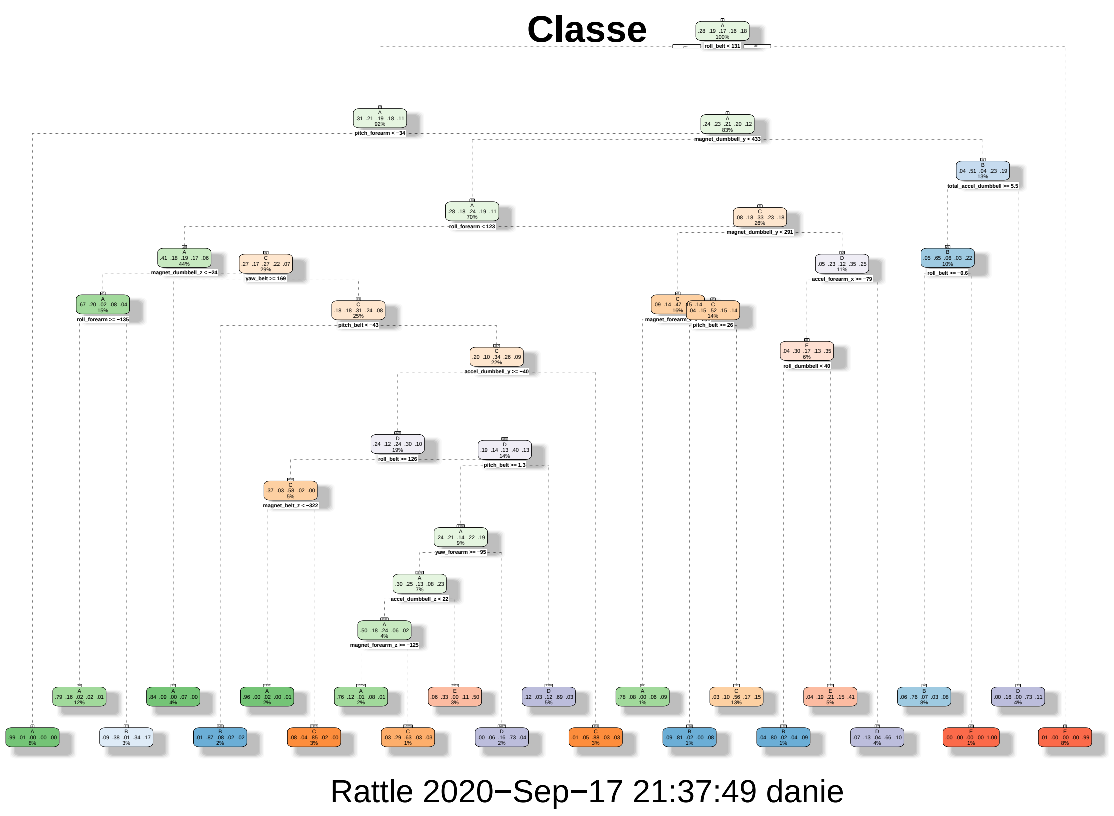
```r
pred_DT <- predict(object = modFit_DT, newdata = testingPC, type = "class")
conMatrix_DT <- confusionMatrix(pred_DT, factor(testingPC$classe))
conMatrix_DT$overall['Accuracy']
```

```
##  Accuracy
## 0.4679694
```

Based on the accuracy of our Decision Tree model we are going to train the same model without the Principal Components.

```r
set.seed(13433)
modFit_DT <- rpart(classe~., data = training, method = "class")
fancyRpartPlot(modFit_DT, main = "Classe")
```

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



```
pred_DT <- predict(object = modFit_DT, newdata = testing, type = "class")
conMatrix_DT <- confusionMatrix(pred_DT, factor(testing$classe))
conMatrix_DT
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1469  170   25   47   14
##          B   74  625   47   73   87
##          C   35   97  819  153  128
##          D   61  104   76  618   60
##          E   35  143   59   73  793
##
## Overall Statistics
##
##                Accuracy : 0.7347
##                  95% CI : (0.7233, 0.746)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6644
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
```

```
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8775   0.5487   0.7982   0.6411   0.7329
## Specificity            0.9392   0.9408   0.9150   0.9388   0.9355
## Pos Pred Value         0.8516   0.6898   0.6648   0.6725   0.7189
## Neg Pred Value         0.9507   0.8968   0.9555   0.9303   0.9396
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2496   0.1062   0.1392   0.1050   0.1347
## Detection Prevalence   0.2931   0.1540   0.2093   0.1562   0.1874
## Balanced Accuracy      0.9084   0.7448   0.8566   0.7900   0.8342
```

Accuracy: 0.7512 and Out-of-sample error: 0.2488

The next models are also going to be trained in the training set without PCA.

Gradient Boosting

```
set.seed(23459)
modFit_GBM <- train(classe~., method="gbm", data = training, verbose=F,
                    trControl = trainControl(method = "cv", number = 3))
pred_GBM <- predict(modFit_GBM, newdata = testing)
conMatrix_GBM <- confusionMatrix(pred_GBM, factor(testing$classe))
conMatrix_GBM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1640   35    0    0    1
##          B   23 1075   32    4   18
##          C    5   28  972   28   10
##          D    3    1   19  927   11
##          E    3    0    3    5 1042
##
## Overall Statistics
##
##                Accuracy : 0.9611
##                  95% CI : (0.9558, 0.9659)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9508
##
##  Mcnemar's Test P-Value : 2.257e-05
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9797   0.9438   0.9474   0.9616   0.9630
## Specificity            0.9915   0.9838   0.9854   0.9931   0.9977
## Pos Pred Value         0.9785   0.9332   0.9319   0.9646   0.9896
## Neg Pred Value         0.9919   0.9865   0.9888   0.9925   0.9917
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2787   0.1827   0.1652   0.1575   0.1771
## Detection Prevalence   0.2848   0.1958   0.1772   0.1633   0.1789
## Balanced Accuracy      0.9856   0.9638   0.9664   0.9774   0.9804
```

Accuracy: 0.9606 and Out-of-sample error: 0.0394

Random Forest

```
set.seed(223345)
modFit_RF <- train(classe~., method="rf", data = training, verboseIter= F,
                   trControl = trainControl(method= "cv", number = 3))
pred_RF <- predict(modFit_RF, newdata = testing)
conMatrix_RF <- confusionMatrix(pred_RF, factor(testing$classe))
conMatrix_RF
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1670    5    0    0    0
##          B    3 1131    3    1    1
##          C    1    3 1017    9    4
##          D    0    0    6  953    1
##          E    0    0    0    1 1076
##
## Overall Statistics
##
##                Accuracy : 0.9935
##                  95% CI : (0.9911, 0.9954)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9918
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9976   0.9930   0.9912   0.9886   0.9945
## Specificity            0.9988   0.9983   0.9965   0.9986   0.9998
## Pos Pred Value         0.9970   0.9930   0.9836   0.9927   0.9991
## Neg Pred Value         0.9990   0.9983   0.9981   0.9978   0.9988
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2838   0.1922   0.1728   0.1619   0.1828
## Detection Prevalence   0.2846   0.1935   0.1757   0.1631   0.1830
## Balanced Accuracy      0.9982   0.9956   0.9939   0.9936   0.9971
```

Accuracy: 0.9941 and Out-of-sample error: 0.0059

Testing the model on the validation set

The best model was the Random Forest. We are going to apply it to the validation set and see the outcome.

```r
predV <- predict(modFit_RF, newdata = validation)
predV_DF <- data.frame(validation$X, predV)
predV_DF
```

```
##    validation.X predV
## 1             1     B
## 2             2     A
## 3             3     B
## 4             4     A
## 5             5     A
## 6             6     E
## 7             7     D
## 8             8     B
## 9             9     A
## 10           10     A
## 11           11     B
## 12           12     C
## 13           13     B
## 14           14     A
## 15           15     E
## 16           16     E
## 17           17     A
## 18           18     B
## 19           19     B
## 20           20     B
```