Practical Machine Learning Report

Prateek Sarangi

08/04/2020

About the dataset.

Context

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. Our goal here 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:
Link:- http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Content

The dataset contains about 160 predictors (most of which are not required) and classifiers column is 'classe' and the exercise pattern is classified into 5 types- A, B, C, D, E.

Acknowledgements

Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013.
Read more: http://groupware.les.inf.puc-rio.br/har#wle__paper__section#ixzz4dPxKFugX

Data cleaning and preprocessing

Removing NA's from training data

```r
training <- read.csv("pml-training.csv")
na_info <- colSums(is.na(training))
na_var <- names((na_info[na_info > 0]))
trainingData <- training[, setdiff(x = names(training), y = na_var)]

dim(trainingData)
```

```
## [1] 19622    93
```

Removing NA's from validation data

```r
validation <- read.csv("pml-testing.csv")
na_info <- colSums(is.na(validation))
na_var <- names((na_info[na_info > 0]))
```

```
validationData <- validation[, setdiff(x = names(validation), y = na_var)]

dim(validationData)
```

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

Removing the column with less contribution to the feature set

```
#For training set
classe <- trainingData$classe
trainRemove <- grepl("^X|timestamp|window", names(trainingData))
trainingData <- trainingData[, !trainRemove]
trainCleaned <- trainingData[, sapply(trainingData, is.numeric)]
trainCleaned$classe <- classe

#For validation set
validationRemove <- grepl("^X|timestamp|window", names(validationData))
validationData <- validationData[, !validationRemove]
validationCleaned <- validationData[, sapply(validationData, is.numeric)]
```

Now we are left with 53 features of training and validation set on which we can train our **Predictive Algorithms**.

Splitting the data into training and test set

We split the dataset into **70%** training and **30%** test set so that we can train our model on the 70% dataset and cross-validate it with the rest 30% of remaining dataset.
For reproducibality I have set the generation of random numbers to be fixed every time the code is executed.

```
smp_size <- floor(0.70 * nrow(trainCleaned))
inTrain <- sample(seq_len(nrow(trainCleaned)), size = smp_size)
trainData <- trainCleaned[inTrain, ]
testData <- trainCleaned[-inTrain, ]
```

Modeling the predictive algorithm

- For this dataset, I will be using **Random Forest**, as this algorithm automatically prioretise the features which are more relevent to the prediction and predict the output accordingly.
- **Random Forest** or **Random Decision Forests** are an **Ensemble Learning** method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction of the individual trees.

```
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```
model <- randomForest(trainData$classe ~ ., data = trainData, ntree = 500, mtry = 6, importance = TRUE)
print(model)
```

```
##
## Call:
##  randomForest(formula = trainData$classe ~ ., data = trainData,      ntree = 500, mtry = 6, importan
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 0.57%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3892    2    1    0    1 0.001026694
## B   16 2630    4    0    0 0.007547170
## C    0   20 2362    3    0 0.009643606
## D    0    0   21 2227    3 0.010661928
## E    0    0    1    6 2546 0.002741872
```

- Then we estimate the erformance of the model on the test set.

```r
predValid <- predict(model, testData, type = "class")
ValAcc <- mean(predValid == testData$classe)*100
sprintf("Validation Accuracy of the model is:- %f", ValAcc)
```

```
## [1] "Validation Accuracy of the model is:- 99.337523"
```

```r
table(predValid,testData$classe)
```

```
##
## predValid    A    B    C    D    E
##         A 1684    2    0    0    0
##         B    0 1140   13    0    0
##         C    0    5 1023   10    0
##         D    0    0    1  955    8
##         E    0    0    0    0 1046
```

Predict the dataset on test data provided

Now, we apply the model to the original testing data set downloaded from the data source.

```r
result <- predict(model, validationCleaned[, -length(names(validationCleaned))])
print(result)
```

```
##  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
```