PML-Course Project

15/03/2020

Introduction

We usually quantify our exercises rather than rate it by quality. In this project, our goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants who 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 This is a project on the Practical Machine Learning course on Coursera where we need to predict the manner in which exercise is performed on a test data set using a sample data called the train data.

Data

The training data for this project are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

The test data are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

Getting and Cleaning Data

We now load the packages useful for this project.

```
library(caret)
```

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

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

```
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:ggplot2':
##
##     margin
```

```
library(parallel)
library(doParallel)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

We now load the data and name the training set as pmltrain and the testing data set as pmltest.

```
pmltrain<-read.csv("D:/R/Practical Machine Learning/pml-training.csv", header=TRUE, na.strings = "NA")
pmltest<-read.csv("D:/R/Practical Machine Learning/pml-testing.csv", header=TRUE)
dim(pmltrain)
```

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

```
dim(pmltest)
```

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

Looking into the data, we can notice that "X", "user_name", "raw_timestamp_part_1", "raw_timestamp_part_2", "cvtd_timestamp", "new_window", "num_window", "roll_belt", "pitch_belt", "yaw_belt" are variables which are irrelevant to our project in prediting the model. So we remove these variables along with this we also remove those variables which has high number of NA's.

Any columns removed on the training datat set must also be removed on the test data set otherwise while predicting the model, prediction will not be proper.

```
#Removing irrelevant variables
pmltrainnew<- pmltrain[,-c(1:10)]
pmltestnew<- pmltest[,-c(1:10)]
dim(pmltrainnew)
```

```
## [1] 19622   150
```

```
dim(pmltestnew)
```

```
## [1]  20 150
```

```
#Identifying rows with NA
na_count <-sapply(pmltrainnew, function(y) sum(length(which(is.na(y)))))
table(na_count)
```

```
## na_count
##     0 19216
##    83    67
```

From the table we can notice that there are columns with either zero missing values or 19216 missing values (or NA). We apply preprocessing in each column if the missing values are less than 75%.

```
#Let us now check if the missing values are less than 75%
unique(colSums(is.na(pmltrainnew))/nrow(pmltrainnew))
```

```
## [1] 0.0000000 0.9793089
```

There are around 67 columns which has 98% more missing values than non-empty entries. Hence, removing those columns is more advisable than applying preprocessing.

```
missing<-sapply(pmltrainnew, function(y) sum(length(which(is.na(y)))))
pmltrainnew<-pmltrainnew[,missing==FALSE]
pmltestnew<-pmltestnew[,missing==FALSE]
dim(pmltrainnew)
```

```
## [1] 19622    83
```

```
dim(pmltestnew)
```

```
## [1] 20 83
```

Using str function on the data set we see that there are a few near zero columns. So we use the nearzerovar function of the caret package to remove those columns.

```
NZV<-nearZeroVar(pmltrainnew)
pmltrainnew<-pmltrainnew[,-NZV]
pmltestnew<-pmltestnew[,-NZV]
```

Partitioning for cross-validation

A better fit of the mdoel can be obtained while applying cross-validation. We use the train set to partition into training and validation set

```
inTrain<- createDataPartition(y=pmltrainnew$classe, p=0.7,list=FALSE)
pmltraining<- pmltrainnew[inTrain,]
pmlvalidation<-pmltrainnew[-inTrain,]
```

Fitting a model

To find the right method to predict the model we utilise methods like- random forest, gbm, lda and combination of methods to predict the model and pick the method with highest accuracy or least error.

Since the data set is huge and calculations can take alot of time, I have utilized parallel computing method while predicting the model. This reduces time drastically and helps is fast computation.

```
#Parallel computing command due to large data set
cluster<-makeCluster(detectCores()-1)
registerDoParallel(cluster)
fitControl<-trainControl(method="cv",
                         number=5,
                         allowParallel = TRUE)

#Fitting a model with rf method and predicting values
modrf<-train(classe~.,data=pmltraining,method="rf", trControl=fitControl)
predrf<-predict(modrf, pmlvalidation)

#Fitting a model with gbm method and predicting values
modgbm<-train(classe~.,data=pmltraining, method="gbm", verbose=FALSE)
predgbm<-predict(modgbm, pmlvalidation)
```

```r
#Fitting a model with lda method and predicting values
modlda<-train(classe~.,data=pmltraining, method="lda")
predlda<-predict(modlda, pmlvalidation)

#Creating data frames for combined models
predrgDF<-data.frame(predrf,predgbm,classe=pmlvalidation$classe)
predglDF<-data.frame(predgbm,predlda,classe=pmlvalidation$classe)
predrlDF<-data.frame(predrf,predlda,classe=pmlvalidation$classe)
predrglDF<-data.frame(predrf,predgbm,predlda,classe=pmlvalidation$classe)

#Combining models and predicting the variables
combrg<-train(classe~.,data=predrgDF)
combgl<-train(classe~.,data=predglDF)
combrl<-train(classe~.,data=predrlDF)
combrgl<-train(classe~.,data=predrglDF)

#stop the cluster used for parallel computation
stopCluster(cluster)
registerDoSEQ()

combpredrg<-predict(combrg,data=predrgDF)
combpredgl<-predict(combgl,data=predglDF)
combpredrl<-predict(combrl,data=predrlDF)
combpredrgl<-predict(combrgl,data=predrglDF)

#Calculating Confusion Matrix for each model to calculate accuracy
confusionMatrix(predrf, pmlvalidation$classe)$overall[1]
```

```
##  Accuracy
## 0.9906542
```

```r
confusionMatrix(predgbm, pmlvalidation$classe)$overall[1]
```

```
##  Accuracy
## 0.9342396
```

```r
confusionMatrix(predlda, pmlvalidation$classe)$overall[1]
```

```
##  Accuracy
## 0.6805438
```

```r
confusionMatrix(combpredrg, pmlvalidation$classe)$overall[1]
```

```
##  Accuracy
## 0.9906542
```

```r
confusionMatrix(combpredgl, pmlvalidation$classe)$overall[1]
```

```
##  Accuracy
## 0.9342396
```

```
confusionMatrix(combpredrl, pmlvalidation$classe)$overall[1]
```

```
##  Accuracy
## 0.9906542
```

```
confusionMatrix(combpredrgl, pmlvalidation$classe)$overall[1]
```

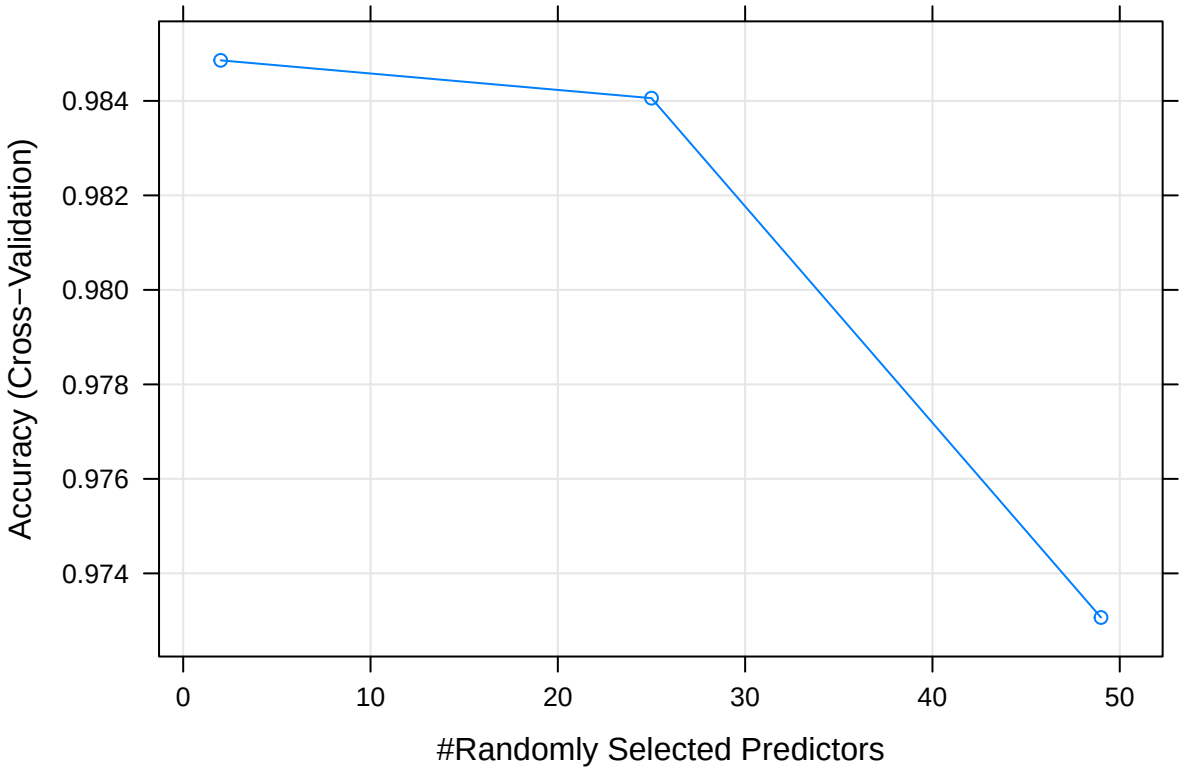
```
##  Accuracy
## 0.9906542
```

From the above models we can see that models using Random Forest method has the highest accuracy which is 98.9%.

```
modrf$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: 2
##
##         OOB estimate of  error rate: 1.08%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3898    4    1    2    1 0.002048131
## B   26 2622   10    0    0 0.013544018
## C    0   30 2365    1    0 0.012938230
## D    0    1   58 2190    3 0.027531083
## E    0    1    2    9 2513 0.004752475
```

```
plot(modrf)
```



The out of bag error is 1.14%. Hence we move forward using the Random Forest method.

Prediction on the test data

Predicting using the Random Forest model:

```
Testpred<-predict(modrf,newdata = pmltest)
Testpred
```

```
##  [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
```

Result above predicts how well (in terms of quality) the exercise was performed by individuals in the test data.