R programming

yeasin parvez

6/25/2020

#Data Loading and Processing

```
library(caret)

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

## Loading required package: lattice

## Loading required package: ggplot2

library(rpart)
library(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 4.0.2

library(RColorBrewer)
library(rattle)

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

## 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

library(corrplot)
```

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

## corrplot 0.84 loaded

library(gbm)

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

## Loaded gbm 2.1.5
```

#Getting, Cleaning and Exploring the data

```
train_in <- read.csv('./pml-training.csv', header=T)
valid_in <- read.csv('./pml-testing.csv', header=T)
dim(train_in)

## [1] 19622   160

dim(valid_in)

## [1]  20 160
```

#Cleaning the input data

```
trainData<- train_in[, colSums(is.na(train_in)) == 0]
validData <- valid_in[, colSums(is.na(valid_in)) == 0]
dim(trainData)

## [1] 19622    93

dim(validData)

## [1] 20 60
```

#We now remove the first seven variables as they have little impact on the outcome classe

```
trainData <- trainData[, -c(1:7)]
validData <- validData[, -c(1:7)]
dim(trainData)

## [1] 19622    86

dim(validData)

## [1] 20 53
```

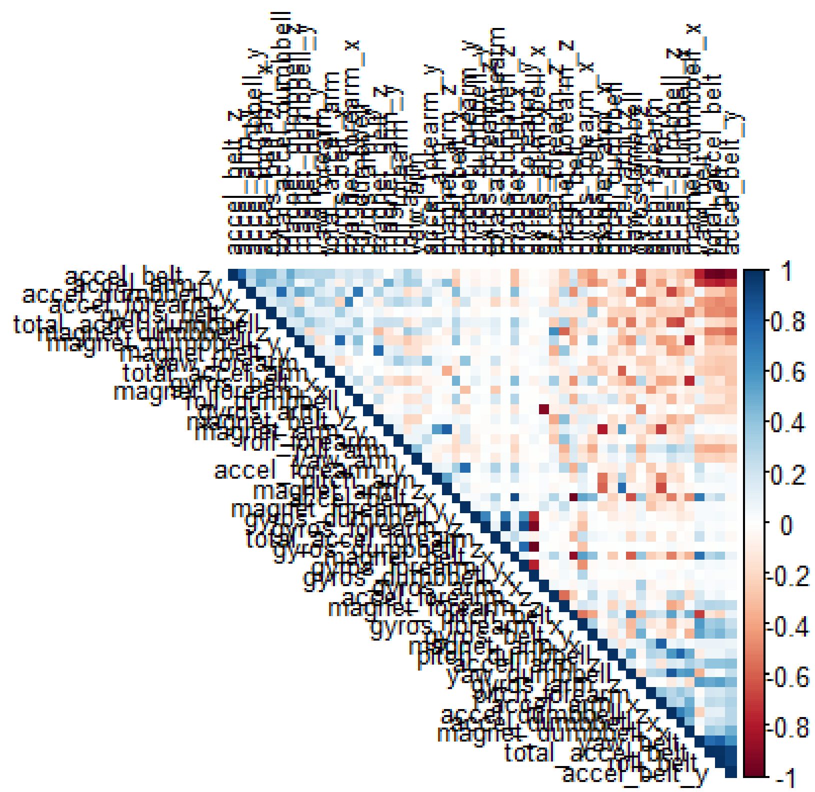
#Preparing the datasets for prediction

```
set.seed(1234)
inTrain <- createDataPartition(trainData$classe, p = 0.7, list = FALSE)
trainData <- trainData[inTrain, ]
testData <- trainData[-inTrain, ]
dim(trainData)

## [1] 13737    86
```

```r
dim(testData)

## [1] 4123   86
```

#Cleaning even further by removing the variables that are near-zero-variance

```r
NZV <- nearZeroVar(trainData)
trainData <- trainData[, -NZV]
testData  <- testData[, -NZV]
dim(trainData)

## [1] 13737    53

dim(testData)

## [1] 4123   53

#After this cleaning we are down now to 49 variables

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



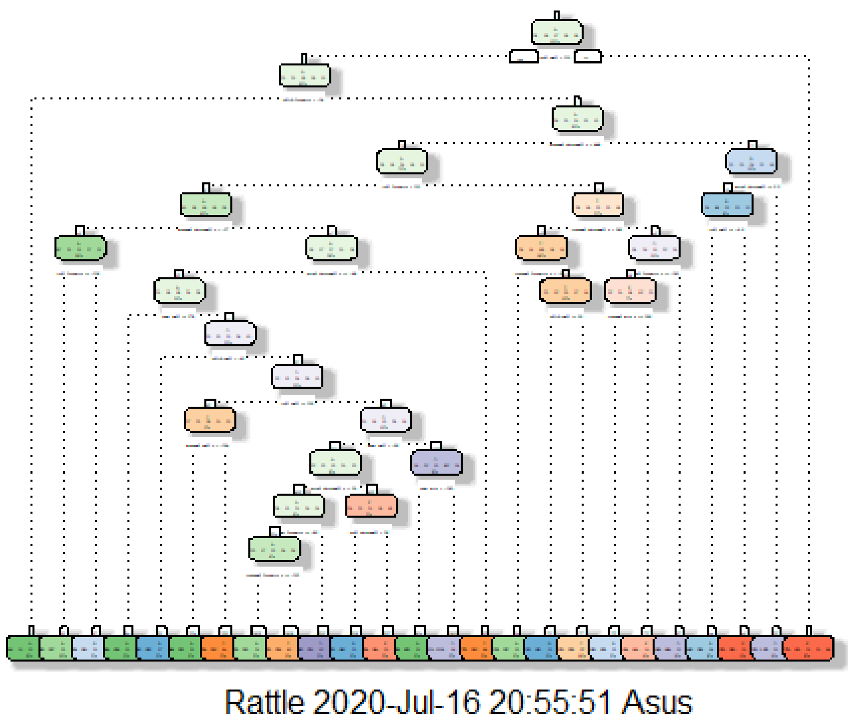
#we use the findCorrelation function to search for highly correlated attributes with a cut off equal to 0.75

```r
highlyCorrelated = findCorrelation(cor_mat, cutoff=0.75)
names(trainData)[highlyCorrelated]#We then obtain the names of highly
correlated attributes
```

```
##  [1] "accel_belt_z"       "roll_belt"          "accel_belt_y"
##  [4] "total_accel_belt"   "accel_dumbbell_z"   "accel_belt_x"
##  [7] "pitch_belt"         "magnet_dumbbell_x" "accel_dumbbell_y"
## [10] "magnet_dumbbell_y" "accel_dumbbell_x"   "accel_arm_x"
## [13] "accel_arm_z"        "magnet_arm_y"       "magnet_belt_z"
## [16] "accel_forearm_y"    "gyros_forearm_y"    "gyros_dumbbell_x"
## [19] "gyros_dumbbell_z"   "gyros_arm_x"
```

#Model building

```
#Prediction with classification trees
set.seed(12345)
decisionTreeMod1<-rpart(classe ~ ., data=trainData, method="class")
fancyRpartPlot(decisionTreeMod1)

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




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

predictTreeMod1 <- predict(decisionTreeMod1, testData, type = "class")
#cmtree <- confusionMatrix(predictTreeMod1, testData$classe)
c#mtree

## function (...)  .Primitive("c")

#plot matrix results
#plot(cmtree$table, col = cmtree$byClass,
```

```
    # main = paste("Decision Tree - Accuracy =",
#round(cmtree$overall['Accuracy'], 4)))
```

#Prediction with Random Forest

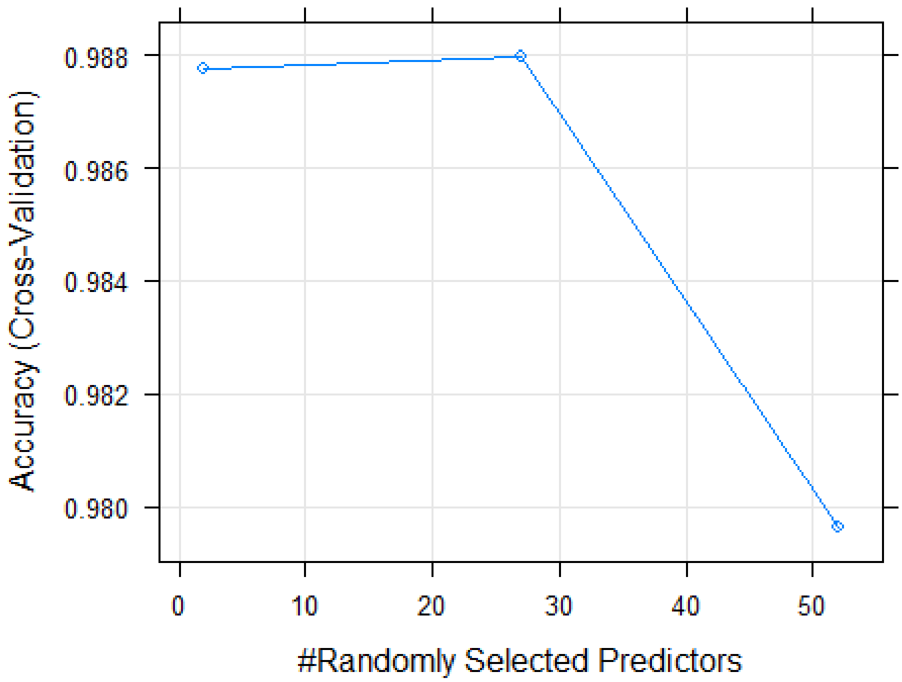
```
controlRF <- trainControl(method="cv", number=3, verboseIter=FALSE)
modRF1 <- train(classe ~ ., data=trainData, method="rf", trControl=controlRF)
modRF1$finalModel

##
## 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: 0.7%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3902    3    0    0    1 0.001024066
## B   19 2634    5    0    0 0.009029345
## C    0   17 2369   10    0 0.011268781
## D    0    1   26 2224    1 0.012433393
## E    0    2    5    6 2512 0.005148515
```

#We then validate the model obtained model “modRF1” on the test data to find out how well it performs by looking at the Accuracy variable

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

#The accuracy rate using the random forest is very high: Accuracy : 1 and therefore the out-of-sample-error is equal to 0***. But it might be due to overfitting.

```
plot(modRF1)
```




```r
#plot(cmrf$table, col = cmrf$byClass, main = paste("Random Forest Confusion Matrix: Accuracy =", round(cmrf$overall['Accuracy'], 4)))
```

#Prediction with Generalized Boosted Regression Models

```r
set.seed(12345)
controlGBM <- trainControl(method = "repeatedcv", number = 5, repeats = 1)
modGBM  <- train(classe ~ ., data=trainData, method = "gbm", trControl =
controlGBM, verbose = FALSE)
modGBM$finalModel

## A gradient boosted model with multinomial loss function.
## 150 iterations were performed.
## There were 52 predictors of which 52 had non-zero influence.

# print model summary
print(modGBM)

## Stochastic Gradient Boosting
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 10990, 10990, 10989, 10991, 10988
## Resampling results across tuning parameters:
```

```
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7521285  0.6858434
##   1                  100      0.8227397  0.7756753
##   1                  150      0.8522224  0.8130469
##   2                   50      0.8564452  0.8181267
##   2                  100      0.9059465  0.8809760
##   2                  150      0.9301168  0.9115592
##   3                   50      0.8969931  0.8695557
##   3                  100      0.9392159  0.9230740
##   3                  150      0.9587251  0.9477728
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth
=
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
#Validate the GBM model
predictGBM <- predict(modGBM, newdata=testData)
#cmGBM <- confusionMatrix(predictGBM, testData$classe)
#cmGBM
```

#Applying the best model to the validation data

```
Results <- predict(modRF1, newdata=validData)
Results
```

```
##  [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
```