An end to end Machine Learning project using XGBoost framework in R studio

A Machine learning approach (done in R-studio) to detect whether a person will have Chronic Kidney Disease ('ckd') nor not ('notckd') based on seleted clinical features.

The raw data in csv format was taken from a GitHub repository of Mr. Abishek Gupta (https://github.com/Elysian01 (https://github.com/Elysian01)). Although deidentified, possibly these are EHR data belonging to real persons. SO, please be respectful and use them in a responsible manner.

Most of the codes and descriptions are re-used from a published R notebook (https://www.kaggle.com/rtatman/machine-learning-with-xgboost-in-r (https://www.kaggle.com/rtatman/machine-learning-with-xgboost-in-r)) written by Rachael Tatman, a Data Scientist at Kaggle.

```
# Importing libraries
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 3.6.3
```

```
library(tidyverse)
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
## Registered S3 method overwritten by 'rvest':
##   method            from
##   read_xml.response xml2
```

```
## -- Attaching packages --------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.1     v purrr   0.3.2
## v tibble  2.1.1     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## Warning: package 'dplyr' was built under R version 3.6.2
```

```
## -- Conflicts ---------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x dplyr::slice()  masks xgboost::slice()
```

```
# Puting the original csv in a data frame
ckd_original <- read_csv("ckd_data.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   id = col_double(),
##   age = col_double(),
##   bp = col_double(),
##   sg = col_double(),
##   al = col_double(),
##   su = col_double(),
##   bgr = col_double(),
##   bu = col_double(),
##   sc = col_double(),
##   sod = col_double(),
##   pot = col_double(),
##   hemo = col_double()
## )
```

```
## See spec(...) for full column specifications.
```

```
head(ckd_original)
```

id <dbl>	age <dbl>	bp <dbl>	sg <dbl>	al <dbl>	su <dbl>	rbc <chr>	pc <chr>	pcc <chr>	ba <chr>
0	48	80	1.020	1	0	*NA*	normal	notpresent	notpresent
1	7	50	1.020	4	0	*NA*	normal	notpresent	notpresent
2	62	80	1.010	2	3	normal	normal	notpresent	notpresent
3	48	70	1.005	4	0	normal	abnormal	present	notpresent
4	51	80	1.010	2	0	normal	normal	notpresent	notpresent
5	60	90	1.015	3	0	*NA*	*NA*	notpresent	notpresent

6 rows | 1-10 of 26 columns

```
# As alaways, we need to run some cleansing operations on the original data set to optimize it for modelling. The cleansing
 process will largely depend on the specific data set and the modelling framework to be used. However, some of the steps are
pretty common and applicable to most of the cases.

# For this particular project we'll need to go through the following cleansing and modification steps:

# 1. Shuffling the rows: A usual and essential prerequisit for training an ML model. The objective is to get rid of any patt
erns in the split datasets. In our original csv file, the rows are sorted on the 'classification' column which contains our
 target variable! As a result, all 'ckd' cases appeared first; this pattern must be removed by shuffling the data frame. Thi
s is a very simple task, we can choose any number as a 'random seed'.

# 2. From the training data set, we need to take out the column containing the target variable: The column named 'classifica
tion' contains our target variable ('ckd'/'notckd')

# 3. For this specific  modelleing framework, we have to make sure that all the data types are in either numeric or logical.
We should convert the character type categorical data into numeric forms by applying an endoding method.

# 3. Converting dataset into testing and training subsets: A common step for ML projects

# 4. Converting the cleaned data frame to a matrix.
```

```
# Shuffle data frame using an arbitrary number as a 'random seed'
set.seed(5523)
ckd_random <- ckd_original[sample(1:nrow(ckd_original)), ]
```

```
# Prepare a subset of the dataframe removing our target variable (contained in 'classification' column) and check the new da
ta frame

ckd_notarget <- ckd_random %>%
    select(-starts_with("classification"))
head(ckd_notarget)
```

id <dbl>	age <dbl>	bp <dbl>	sg <dbl>	al <dbl>	su <dbl>	rbc <chr>	pc <chr>	pcc <chr>	ba <chr>
18	60	100	1.025	0	3	*NA*	normal	notpresent	notpresent
318	61	70	1.025	0	0	normal	normal	notpresent	notpresent
207	50	70	1.010	0	0	*NA*	normal	notpresent	notpresent
182	61	80	1.020	0	0	*NA*	normal	notpresent	notpresent
173	17	70	1.015	1	0	abnormal	normal	notpresent	notpresent
153	55	90	1.010	2	1	abnormal	abnormal	notpresent	notpresent

6 rows | 1-10 of 25 columns

```
# We have succesfully removed the classification column. However, we'll need the classification labels to train and evaluate
the models.So, before we forget, this is a good time to generate a new vector containing the target labels ('ckd' or 'notck
d'). We'll check the new vector by the head()function.

ckd_labels <- ckd_random[, c("classification")]
head(ckd_labels)
```

classification <chr>
ckd

classification <chr>
notckd
ckd
ckd
ckd
ckd

6 rows

```
# Fantastic! We got the labels in the same sorted order as in the parent data frame (ckd_random). However, since the data ty
pe is 'character'and we need to change that to either a 'numeric' or a 'logical' data type for the modelling framework(XGBoo
st). So, let's convert it to a boolean vector and check.

ckd_labels_boolean <- with(ckd_labels, ifelse(classification == 'ckd', TRUE, ifelse(classification == 'notckd', FALSE, NA)))

head (ckd_labels_boolean)
```

```
## [1]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
```

```
# Now let us take a closer look on the the structure of our data frame before proceeding to build our model
str(ckd_notarget)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    400 obs. of  25 variables:
##  $ id   : num  18 318 207 182 173 153 36 222 346 34 ...
##  $ age  : num  60 61 50 61 17 55 76 74 33 70 ...
##  $ bp   : num  100 70 70 80 70 90 70 60 60 70 ...
##  $ sg   : num  1.02 1.02 1.01 1.02 1.01 ...
##  $ al   : num  0 0 0 0 1 2 1 NA NA 1 ...
##  $ su   : num  3 0 0 0 0 1 0 NA NA 0 ...
##  $ rbc  : chr  NA "normal" NA NA ...
##  $ pc   : chr  "normal" "normal" "normal" "normal" ...
##  $ pcc  : chr  "notpresent" "notpresent" "notpresent" "notpresent" ...
##  $ ba   : chr  "notpresent" "notpresent" "notpresent" "notpresent" ...
##  $ bgr  : num  263 120 230 131 22 273 92 108 130 171 ...
##  $ bu   : num  27 29 50 23 1.5 235 29 68 41 153 ...
##  $ sc   : num  1.3 0.7 2.2 0.8 7.3 14.2 1.8 1.8 0.9 5.2 ...
##  $ sod  : num  135 137 NA 140 145 132 133 NA 141 NA ...
##  $ pot  : num  4.3 3.5 NA 4.1 2.8 3.4 3.9 NA 4.4 NA ...
##  $ hemo : num  12.7 17.4 12 11.3 13.1 8.3 10.3 NA 15.5 NA ...
##  $ pcv  : chr  "37" "52" "41" "35" ...
##  $ wc   : chr  "11400" "7000" "10400" NA ...
##  $ rc   : chr  "4.3" "5.3" "4.6" NA ...
##  $ htn  : chr  "yes" "no" "yes" "no" ...
##  $ dm   : chr  "yes" "no" "yes" "no" ...
##  $ cad  : chr  "yes" "no" "no" "no" ...
##  $ appet: chr  "good" "good" "good" "good" ...
##  $ pe   : chr  "no" "no" "no" "no" ...
##  $ ane  : chr  "no" "no" "no" "no" ...
```

```
# Oh no! In the columns named 'pcv', 'wc', and 'rc', the  numeric data are stored as charater. We need to fix that by conver
ting them into numeric data

ckd_notarget$pcv = as.numeric(as.character(ckd_notarget$pcv))
```

```
## Warning: NAs introduced by coercion
```

```
ckd_notarget$wc = as.numeric(as.character(ckd_notarget$wc))
```

```
## Warning: NAs introduced by coercion
```

```
ckd_notarget$rc = as.numeric(as.character(ckd_notarget$rc))
```

```
## Warning: NAs introduced by coercion
```

```
# Rechecking the ckd_notarget df after the correction (strings to numeric, for simplicity no new data frame is created)

str(ckd_notarget)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    400 obs. of  25 variables:
##  $ id   : num  18 318 207 182 173 153 36 222 346 34 ...
##  $ age  : num  60 61 50 61 17 55 76 74 33 70 ...
##  $ bp   : num  100 70 70 80 70 90 70 60 60 70 ...
##  $ sg   : num  1.02 1.02 1.01 1.02 1.01 ...
##  $ al   : num  0 0 0 0 1 2 1 NA NA 1 ...
##  $ su   : num  3 0 0 0 0 1 0 NA NA 0 ...
##  $ rbc  : chr  NA "normal" NA NA ...
##  $ pc   : chr  "normal" "normal" "normal" "normal" ...
##  $ pcc  : chr  "notpresent" "notpresent" "notpresent" "notpresent" ...
##  $ ba   : chr  "notpresent" "notpresent" "notpresent" "notpresent" ...
##  $ bgr  : num  263 120 230 131 22 273 92 108 130 171 ...
##  $ bu   : num  27 29 50 23 1.5 235 29 68 41 153 ...
##  $ sc   : num  1.3 0.7 2.2 0.8 7.3 14.2 1.8 1.8 0.9 5.2 ...
##  $ sod  : num  135 137 NA 140 145 132 133 NA 141 NA ...
##  $ pot  : num  4.3 3.5 NA 4.1 2.8 3.4 3.9 NA 4.4 NA ...
##  $ hemo : num  12.7 17.4 12 11.3 13.1 8.3 10.3 NA 15.5 NA ...
##  $ pcv  : num  37 52 41 35 41 22 32 NA 52 NA ...
##  $ wc   : num  11400 7000 10400 NA 11200 14600 NA NA 4300 NA ...
##  $ rc   : num  4.3 5.3 4.6 NA NA 2.9 NA NA 5.8 NA ...
##  $ htn  : chr  "yes" "no" "yes" "no" ...
##  $ dm   : chr  "yes" "no" "yes" "no" ...
##  $ cad  : chr  "yes" "no" "no" "no" ...
##  $ appet: chr  "good" "good" "good" "good" ...
##  $ pe   : chr  "no" "no" "no" "no" ...
##  $ ane  : chr  "no" "no" "no" "no" ...
```

```
# Great! We fixed the data type error. Note that some of the values could not be converted and they are replaced by null val
ues (NA).

# Now, we'll change the remaining character type data into numeric by one-hot encoding. 'caret' is a powerful library that w
ill do this transformation with two lines of codes (find the details here: https://www.pluralsight.com/guides/encoding-data-
with-r).

# We also need to remove  the 'id' column which is numeric but it would make no sense if included in our model.

# we'll nake a new data frame named ckd_transformed after the modifications and have a look at it using the glimpse() functi
on

library(caret)
```

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

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
dmy <- dummyVars(" ~ .", data = ckd_notarget, fullRank = T)
ckd_transformed <- data.frame(predict(dmy, newdata = ckd_notarget))

ckd_transformed <- select(ckd_transformed, -c(id))

glimpse(ckd_transformed)
```

```
## Observations: 400
## Variables: 24
## $ age        <dbl> 60, 61, 50, 61, 17, 55, 76, 74, 33, 70, 52, 73, NA,...
## $ bp         <dbl> 100, 70, 70, 80, 70, 90, 70, 60, 60, 70, 80, 80, 80...
## $ sg         <dbl> 1.025, 1.025, 1.010, 1.020, 1.015, 1.010, 1.015, NA...
## $ al         <dbl> 0, 0, 0, 0, 1, 2, 1, NA, NA, 1, 0, 2, NA, NA, 0, 0,...
## $ su         <dbl> 3, 0, 0, 0, 0, 1, 0, NA, NA, 0, 0, 0, NA, NA, 0, 0,...
## $ rbcnormal  <dbl> NA, 1, NA, NA, 0, 0, 1, NA, 1, 1, 1, 0, NA, NA, 1, ...
## $ pcnormal   <dbl> 1, 1, 1, 1, 1, 0, 1, NA, 1, NA, 1, 0, NA, NA, 1, 1,...
## $ pccpresent <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ...
## $ bapresent  <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ...
## $ bgr        <dbl> 263, 120, 230, 131, 22, 273, 92, 108, 130, 171, 99,...
## $ bu         <dbl> 27.0, 29.0, 50.0, 23.0, 1.5, 235.0, 29.0, 68.0, 41....
## $ sc         <dbl> 1.3, 0.7, 2.2, 0.8, 7.3, 14.2, 1.8, 1.8, 0.9, 5.2, ...
## $ sod        <dbl> 135.0, 137.0, NA, 140.0, 145.0, 132.0, 133.0, NA, 1...
## $ pot        <dbl> 4.3, 3.5, NA, 4.1, 2.8, 3.4, 3.9, NA, 4.4, NA, 3.7,...
## $ hemo       <dbl> 12.7, 17.4, 12.0, 11.3, 13.1, 8.3, 10.3, NA, 15.5, ...
## $ pcv        <dbl> 37, 52, 41, 35, 41, 22, 32, NA, 52, NA, 52, 33, 53,...
## $ wc         <dbl> 11400, 7000, 10400, NA, 11200, 14600, NA, NA, 4300,...
## $ rc         <dbl> 4.3, 5.3, 4.6, NA, NA, 2.9, NA, NA, 5.8, NA, 5.3, 4...
## $ htnyes     <dbl> 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, ...
## $ dmyes      <dbl> 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, ...
## $ cadyes     <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, ...
## $ appetpoor  <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ...
## $ peyes      <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ aneyes     <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
```

```
# Rechek some of the rows, let's check the first 10 rows:
head(ckd_transformed, n=10)
```

	age <dbl>	bp <dbl>	sg <dbl>	al <dbl>	su <dbl>	rbcnormal <dbl>	pcnormal <dbl>	pccpresent <dbl>	bapresent <dbl>
1	60	100	1.025	0	3	*NA*	1	0	0
2	61	70	1.025	0	0	1	1	0	0
3	50	70	1.010	0	0	*NA*	1	0	0

	age <dbl>	bp <dbl>	sg <dbl>	al <dbl>	su <dbl>	rbcnormal <dbl>	pcnormal <dbl>	pccpresent <dbl>	bapresent <dbl>
4	61	80	1.020	0	0	*NA*	1	0	0
5	17	70	1.015	1	0	0	1	0	0
6	55	90	1.010	2	1	0	0	0	0
7	76	70	1.015	1	0	1	1	0	0
8	74	60	*NA*	*NA*	*NA*	*NA*	*NA*	0	0
9	33	60	*NA*	*NA*	*NA*	1	1	0	0
10	70	70	1.010	1	0	1	*NA*	1	1

1-10 of 10 rows | 1-10 of 25 columns

```
# Finally, everything looks good. We are now ready to convert the data frame into a matrix to start modelling!

ckd_matrix <- data.matrix(ckd_transformed)
```

```
head(ckd_matrix)
```

```
##   age  bp    sg al su rbcnormal pcnormal pccpresent bapresent bgr    bu
## 1  60 100 1.025  0  3        NA        1          0         0 263  27.0
## 2  61  70 1.025  0  0         1        1          0         0 120  29.0
## 3  50  70 1.010  0  0        NA        1          0         0 230  50.0
## 4  61  80 1.020  0  0        NA        1          0         0 131  23.0
## 5  17  70 1.015  1  0         0        1          0         0  22   1.5
## 6  55  90 1.010  2  1         0        0          0         0 273 235.0
##     sc sod pot hemo pcv    wc  rc htnyes dmyes cadyes appetpoor peyes
## 1  1.3 135 4.3 12.7  37 11400 4.3      1     1      1         0     0
## 2  0.7 137 3.5 17.4  52  7000 5.3      0     0      0         0     0
## 3  2.2  NA  NA 12.0  41 10400 4.6      1     1      0         0     0
## 4  0.8 140 4.1 11.3  35    NA  NA      0     0      0         0     0
## 5  7.3 145 2.8 13.1  41 11200  NA      0     0      0         0     0
## 6 14.2 132 3.4  8.3  22 14600 2.9      1     1      0         1     1
##   aneyes
## 1      0
## 2      0
## 3      0
## 4      0
## 5      0
## 6      1
```

```
# Now we'll devide our matrix objects (both 'ckd_matix' and 'ckd_labels_boolean') into training (with 70% of the total avail
able rows) and testing (the remaining 30% of rows) subsets.


rows_training <- round(length(ckd_labels_boolean) * .7) # this is to find the number of rows to be used in the training subs
et

train_data <- ckd_matrix[1:rows_training,]
train_labels <- ckd_labels_boolean[1:rows_training]

# testing data
test_data <- ckd_matrix[-(1:rows_training),]
test_labels <- ckd_labels_boolean[-(1:rows_training)]
```

```
# For a faster operation of the XGBoost framework, we'll convert our data matrix into 'Dmatrix' objects. This is an optional step.


dtrain <- xgb.DMatrix(data = train_data, label= train_labels)
dtest <- xgb.DMatrix(data = test_data, label= test_labels)
```

```
# We'll train our models using binary logistic regression (because our goal is to predict something that is binary in natur
e. Either patients are going to have ckd or they do not). Please note that if we do not specify binarry logistic as our obje
ctive function, a linear regression will be conducted by XGBoost.


model <- xgboost(data = dtrain,
                 nround = 16, # this is the maximum number of boosting iterations, you can use a different number
                 objective = "binary:logistic")
```

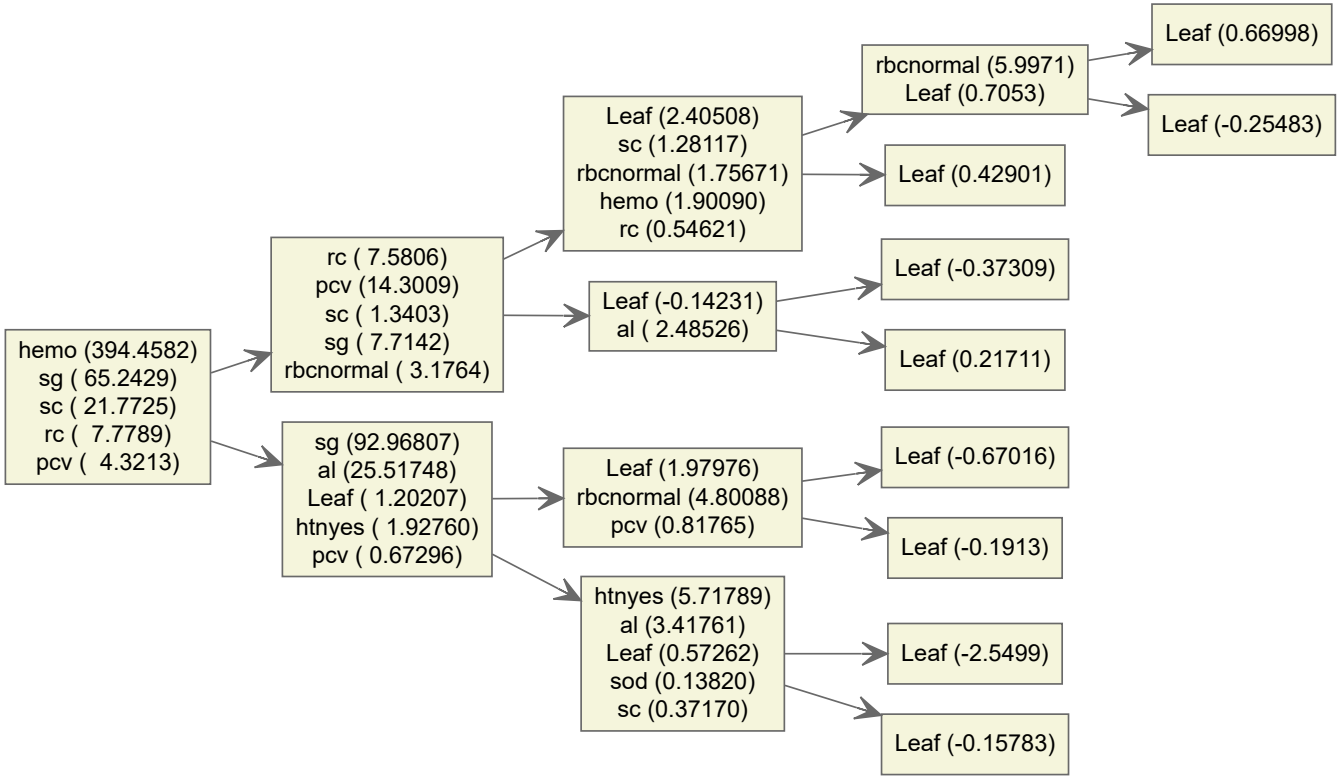
```
## [09:12:18] WARNING: amalgamation/../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used wi
th the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restor
e the old behavior.
## [1]  train-logloss:0.463159
## [2]  train-logloss:0.329585
## [3]  train-logloss:0.244143
## [4]  train-logloss:0.180349
## [5]  train-logloss:0.138961
## [6]  train-logloss:0.106278
## [7]  train-logloss:0.083863
## [8]  train-logloss:0.067857
## [9]  train-logloss:0.055248
## [10] train-logloss:0.045481
## [11] train-logloss:0.038451
## [12] train-logloss:0.031885
## [13] train-logloss:0.026582
## [14] train-logloss:0.023639
## [15] train-logloss:0.021345
## [16] train-logloss:0.018762
```

```
# The error on the training data is depiccted by 'logloss'. We can see that the loss or error is reduced gradually from roun
d 1 to 16.

# Now the real test! Let's see how our model performs in terms of making predictions in the test data set
pred <- predict(model, dtest)

# View the error
err <- mean(as.numeric(pred > 0.5) != test_labels)
print(paste("test-error=", err))
```

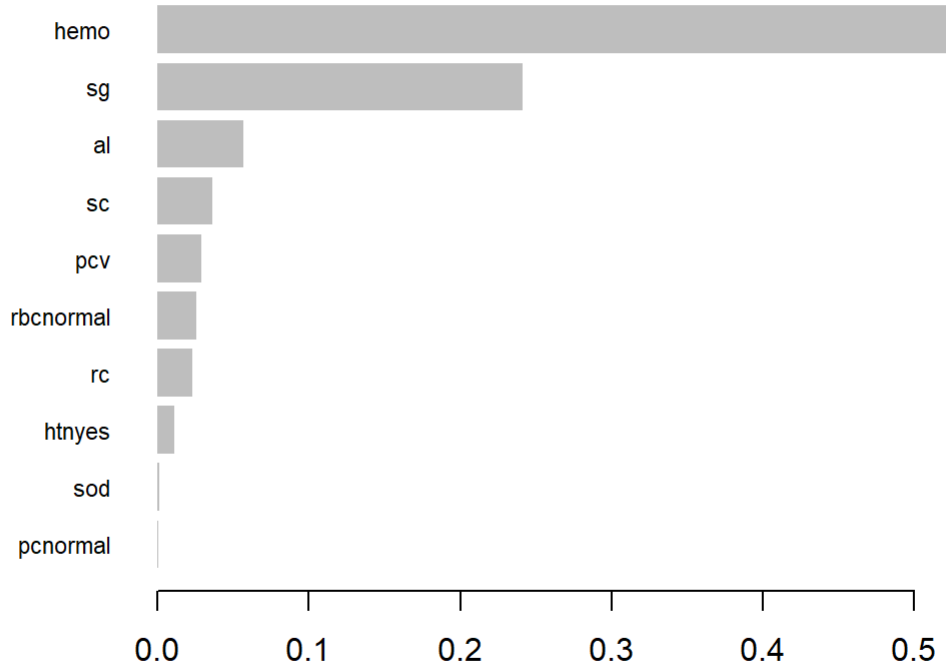
```
## [1] "test-error= 0.00833333333333333"
```

```
#Bravo! We see a lower error on our testing data compared to the training data.It implies that out model is not over-fitted.

# Now let's do some plotting to vizualize the trees from the model

xgb.plot.multi.trees(feature_names = names(ckd_matrix),
                     model = model)
```

```
## Column 2 ['No'] of item 2 is missing in item 1. Use fill=TRUE to fill with NA (NULL for list columns), or use.names=FALSE
to ignore column names. use.names='check' (default from v1.12.2) emits this message and proceeds as if use.names=FALSE for
backwards compatibility. See news item 5 in v1.12.2 for options to control this message.
```




```
# And finally, another plotting to visualize the relative importance of the features in our model
importance_matrix <- xgb.importance(names(ckd_matrix), model = model)
xgb.plot.importance(importance_matrix)
```




```
# The basics are done. However, we can always try to improve our model performance by tuning it.But that is another story fo
r another day!
```