NearDuplicate

DeboraReis

16/07/2017

```
# Carrega os dados de dicionario de dados de tabelas e colunas
library(readr)
dic <- read_delim("~/Similares/db2ExportFull.csv",  ";", escape_double = FALSE, trim_ws = TRUE)
dic = as.data.frame(dic[,1:15])
```

PROCESSAMENTO E TRANSFORMACAO

```
# Pega só os que tem comentarios como dicionario de dados
dic = na.omit(dic)

# Filtra server
dic = dic[dic$HOST_NAME == '[redacted]' | dic$HOST_NAME == '[redacted]' | dic$HOST_NAME == '[redacted]'
          | dic$HOST_NAME == '[redacted]' | dic$HOST_NAME == '[redacted]',]
table(dic$HOST_NAME)
```

```
##
## [redacted]
##     1457       117       515      1171      1046
```

```
#[redacted]
#    8658      1457      8385       117       515      1171      1046  # Colunas
#      35         3        11         4         1         2         3  # Schema
#     829       112       584        26        46       155       146  # Tabela
#     252       177       226       118       137       188       249  # Max nchar de comment de tabela
#     254       253       250       181       254      250        244  # Max nchar de comment de coluna


# Qtos Schemas possuem dicionario de dados de tabelas # 60
length(unique(paste(dic$HOST_NAME, dic$TABSCHEMA, sep = " ")))
```

```
## [1] 13
```

```
length(unique(dic$TABSCHEMA))
```

```
## [1] 13
```

```
# Qtos Tabelas possuem dicionario de dados de tabelas # 2.347
length(unique(paste(dic$HOST_NAME, dic$TABSCHEMA, dic$TABNAME, sep = " ")))
```

```
## [1] 589
```

```
# Qtas Colunas possuem dicionario de dados de colunas # 21.521
length(unique(paste(dic$HOST_NAME, dic$TABSCHEMA, dic$TABNAME, dic$COLNAME, sep = " ")))
```

```
## [1] 4306
```

```
# Qtos diferentes valores de comentario de tabela # 1.881
length(unique(dic$TABLE_COMENT))
```

```
## [1] 484
```

```r
# Qtos diferentes valores de comentario de coluna # 13.646
length(unique(dic$COLUMN_COMENTS))
```

```
## [1] 3332
```

```r
# Faz tratamento do texto no nome do schema, tabela, coluna e tipo
dic$TABSCHEMA = gsub('[[:punct:]]', '', dic$TABSCHEMA)
dic$TABSCHEMA = gsub('\\"', '', dic$TABSCHEMA)
dic$TABSCHEMA = gsub('\\?', '', dic$TABSCHEMA)
dic$TABSCHEMA = gsub('ã', 'a', dic$TABSCHEMA)
dic$TABSCHEMA = gsub('õ', 'o', dic$TABSCHEMA)
dic$TABSCHEMA = gsub('ç', 'c', dic$TABSCHEMA)
dic$TABSCHEMA = gsub('à', 'a', dic$TABSCHEMA)
dic$TABSCHEMA = gsub('á', 'a', dic$TABSCHEMA)
dic$TABSCHEMA = gsub('é', 'e', dic$TABSCHEMA)
dic$TABSCHEMA = gsub('í', 'i', dic$TABSCHEMA)
dic$TABSCHEMA = gsub('ó', 'o', dic$TABSCHEMA)
dic$TABSCHEMA = gsub('ú', 'u', dic$TABSCHEMA)
dic$TABSCHEMA = tolower(dic$TABSCHEMA)
dic$TABSCHEMA= gsub("[^\x20-\x7E]", "", dic$TABSCHEMA)

dic$TABNAME = gsub('[[:punct:]]', '', dic$TABNAME)
dic$TABNAME = gsub('\\"', '', dic$TABNAME)
dic$TABNAME = gsub('\\?', '', dic$TABNAME)
dic$TABNAME = gsub('ã', 'a', dic$TABNAME)
dic$TABNAME = gsub('õ', 'o', dic$TABNAME)
dic$TABNAME = gsub('ç', 'c', dic$TABNAME)
dic$TABNAME = gsub('à', 'a', dic$TABNAME)
dic$TABNAME = gsub('á', 'a', dic$TABNAME)
dic$TABNAME = gsub('é', 'e', dic$TABNAME)
dic$TABNAME = gsub('í', 'i', dic$TABNAME)
dic$TABNAME = gsub('ó', 'o', dic$TABNAME)
dic$TABNAME = gsub('ú', 'u', dic$TABNAME)
dic$TABNAME = tolower(dic$TABNAME)
dic$TABNAME = gsub("[^\x20-\x7E]", "", dic$TABNAME)

dic$COLNAME = gsub('[[:punct:]]', '', dic$COLNAME)
dic$COLNAME = gsub('\\"', '', dic$COLNAME)
dic$COLNAME = gsub('\\?', '', dic$COLNAME)
dic$COLNAME = gsub('ã', 'a', dic$COLNAME)
dic$COLNAME = gsub('õ', 'o', dic$COLNAME)
dic$COLNAME = gsub('ç', 'c', dic$COLNAME)
dic$COLNAME = gsub('à', 'a', dic$COLNAME)
dic$COLNAME = gsub('á', 'a', dic$COLNAME)
dic$COLNAME = gsub('é', 'e', dic$COLNAME)
dic$COLNAME = gsub('í', 'i', dic$COLNAME)
dic$COLNAME = gsub('ó', 'o', dic$COLNAME)
dic$COLNAME = gsub('ú', 'u', dic$COLNAME)
dic$COLNAME = tolower(dic$COLNAME)
dic$COLNAME = gsub("[^\x20-\x7E]", "", dic$COLNAME)

dic$TYPENAME = gsub('[[:punct:]]', '', dic$TYPENAME)
dic$TYPENAME = gsub('\\"', '', dic$TYPENAME)
dic$TYPENAME = gsub('\\?', '', dic$TYPENAME)
```

```r
dic$TYPENAME = gsub('ã', 'a', dic$TYPENAME)
dic$TYPENAME = gsub('õ', 'o', dic$TYPENAME)
dic$TYPENAME = gsub('ç', 'c', dic$TYPENAME)
dic$TYPENAME = gsub('à', 'a', dic$TYPENAME)
dic$TYPENAME = gsub('á', 'a', dic$TYPENAME)
dic$TYPENAME = gsub('é', 'e', dic$TYPENAME)
dic$TYPENAME = gsub('í', 'i', dic$TYPENAME)
dic$TYPENAME = gsub('ó', 'o', dic$TYPENAME)
dic$TYPENAME = gsub('ú', 'u', dic$TYPENAME)
dic$TYPENAME = tolower(dic$TYPENAME)
dic$TYPENAME = gsub("[^\x20-\x7E]", "", dic$TYPENAME)

# Faz tratamento no texto das colunas de comentario de tabela e de coluna
dic$TABLE_COMENT = gsub('[[:punct:]]', '', dic$TABLE_COMENT)
dic$TABLE_COMENT = gsub('\\"', '', dic$TABLE_COMENT)
dic$TABLE_COMENT = gsub('\\?', '', dic$TABLE_COMENT)
dic$TABLE_COMENT = gsub('ã', 'a', dic$TABLE_COMENT)
dic$TABLE_COMENT = gsub('õ', 'o', dic$TABLE_COMENT)
dic$TABLE_COMENT = gsub('ç', 'c', dic$TABLE_COMENT)
dic$TABLE_COMENT = gsub('à', 'a', dic$TABLE_COMENT)
dic$TABLE_COMENT = gsub('á', 'a', dic$TABLE_COMENT)
dic$TABLE_COMENT = gsub('é', 'e', dic$TABLE_COMENT)
dic$TABLE_COMENT = gsub('í', 'i', dic$TABLE_COMENT)
dic$TABLE_COMENT = gsub('ó', 'o', dic$TABLE_COMENT)
dic$TABLE_COMENT = gsub('ú', 'u', dic$TABLE_COMENT)
dic$TABLE_COMENT = tolower(dic$TABLE_COMENT)
dic$TABLE_COMENT = gsub("[^\x20-\x7E]", "", dic$TABLE_COMENT)

dic$COLUMN_COMENTS = gsub('[[:punct:]]', '', dic$COLUMN_COMENTS)
dic$COLUMN_COMENTS = gsub('\\"', '', dic$COLUMN_COMENTS)
dic$COLUMN_COMENTS = gsub('\\?', '', dic$COLUMN_COMENTS)
dic$COLUMN_COMENTS = gsub('ã', 'a', dic$COLUMN_COMENTS)
dic$COLUMN_COMENTS = gsub('õ', 'o', dic$COLUMN_COMENTS)
dic$COLUMN_COMENTS = gsub('ç', 'c', dic$COLUMN_COMENTS)
dic$COLUMN_COMENTS = gsub('à', 'a', dic$COLUMN_COMENTS)
dic$COLUMN_COMENTS = gsub('á', 'a', dic$COLUMN_COMENTS)
dic$COLUMN_COMENTS = gsub('é', 'e', dic$COLUMN_COMENTS)
dic$COLUMN_COMENTS = gsub('í', 'i', dic$COLUMN_COMENTS)
dic$COLUMN_COMENTS = gsub('ó', 'o', dic$COLUMN_COMENTS)
dic$COLUMN_COMENTS = gsub('ú', 'u', dic$COLUMN_COMENTS)
dic$COLUMN_COMENTS = tolower(dic$COLUMN_COMENTS)
dic$COLUMN_COMENTS <- gsub("[^\x20-\x7E]", "", dic$COLUMN_COMENTS)

# Cria o id
dic$id = paste(dic$HOST_NAME, dic$TABSCHEMA, dic$TABNAME, dic$COLNAME, sep = " ")
```

```r
# Compara par a par todos os campos
library(dplyr)
a = expand.grid(dic$id, dic$id)
b = expand.grid(dic$HOST_NAME, dic$HOST_NAME)
c = expand.grid(dic$TABSCHEMA, dic$TABSCHEMA)
e = expand.grid(dic$TABNAME, dic$TABNAME)
f = expand.grid(dic$COLNAME, dic$COLNAME)
g = expand.grid(dic$TYPENAME, dic$TYPENAME)
```

```r
h = expand.grid(dic$TABLE_COMENT, dic$TABLE_COMENT)
i = expand.grid(dic$COLUMN_COMENTS, dic$COLUMN_COMENTS)
j = expand.grid(dic$QTDA_LINHAS, dic$QTDA_LINHAS)
l = expand.grid(dic$LENGTH, dic$LENGTH)
m = expand.grid(dic$NULLS, dic$NULLS)
n = expand.grid(dic$IS_FK_COLUMN, dic$IS_FK_COLUMN)
o = expand.grid(dic$LASTUSED, dic$LASTUSED)
p = expand.grid(dic$TAMANHO, dic$TAMANHO)
columns = bind_cols(a,b,c,e,f,g,h,i,j,l,m,n,o,p)
rm(a,b,c,e,f,g,h,i,j,l,m,n,o,p,dic)
gc()
```

```
##            used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells    578793   31.0    1168576   62.5    940480   50.3
## Vcells 260818771 1989.9  400428314 3055.1 396731914 3026.9
```

```r
# Retira da combinação de pares a combinação com ele mesmo e a combinação inversa
columns = as.data.frame(columns)
columns = columns[!duplicated(apply(columns,1,function(x) paste(sort(x),collapse=''))),]

# Renomeia os nomes das colunas
colnames(columns) = c("id2", "id1", "server2", "server1", "schema2", "schema1", "table2", "table1",
                "col2", "col1", "type2", "type1", "dicTab2", "dicTab1", "dicCol2", "dicCol1",
                "nrow2", "nrow1", "lengthcol2", "lengthcol1", "null2", "null1",
                "fk2", "fk1", "lastused2", "lastused1", "size2", "size1")

#reordena as colunas
columns = columns[,c("id1", "id2", "server1", "server2", "schema1", "schema2", "table1", "table2",
                "col1", "col2", "type1", "type2", "dicTab1", "dicTab2", "dicCol1", "dicCol2",
                "nrow1", "nrow2", "lengthcol1", "lengthcol2", "null1", "null2",
                "fk1", "fk2", "lastused1", "lastused2", "size1", "size2")]

# Retira a comparacao com ele mesmo
library(dplyr)
columns = columns %>% filter(id1 != id2)

# Retira os que tem dicionario "realizada uma carga do datasatge tabela tbpessoa" que representa 406 ob
columns = columns[columns$dicCol1 != "realizada uma carga do datasatge tabela tbpessoa",]
columns = columns[columns$dicCol2 != "realizada uma carga do datasatge tabela tbpessoa",]

# Salva
write.csv2(x = columns, file = "columns.csv", row.names = TRUE)
```

```r
# Carrega os dados
columns = read_delim("~/Similares/columns.csv", ";", escape_double = FALSE, trim_ws = TRUE)
columns = as.data.frame(columns[,2:29])

# Calcula distancia das variaveis tipo texto com Levenshtein e Cosseno
library(stringdist)
columns$lv_schema = stringdist(columns$schema1, columns$schema2, method = "lv")
columns$cos_schema = stringdist(columns$schema1, columns$schema2, method = "cosine")
columns$lv_tab = stringdist(columns$table1, columns$table2, method = "lv")
columns$cos_tab = stringdist(columns$table1, columns$table2, method = "cosine")
columns$lv_col = stringdist(columns$col1, columns$col2, method = "lv")
```

```r
columns$cos_col = stringdist(columns$col1, columns$col2, method = "cosine")
columns$lv_type = stringdist(columns$type1, columns$type2, method = "lv")
columns$cos_type = stringdist(columns$type1, columns$type2, method = "cosine")
columns$lv_dicTab = stringdist(columns$dicTab1, columns$dicTab2, method = "lv")
columns$cos_dicTab = stringdist(columns$dicTab1, columns$dicTab2, method = "cosine")
columns$lv_dicCol = stringdist(columns$dicCol1, columns$dicCol2, method = "lv")
columns$cos_dicCol = stringdist(columns$dicCol1, columns$dicCol2, method = "cosine")
columns$lv_null = stringdist(columns$null1, columns$null2, method = "lv")
columns$cos_null = stringdist(columns$null1, columns$null2, method = "cosine")
columns$lv_fk = stringdist(columns$fk1, columns$fk2, method = "lv")
columns$cos_fk = stringdist(columns$fk1, columns$fk2, method = "cosine")

# Calcula a distancia entre as variaveis numericas
columns$dist_nrow = apply(columns[,c('nrow1','nrow2')], 1, function(x) sd(x))
columns$dist_length = apply(columns[,c('lengthcol1','lengthcol2')], 1, function(x) sd(x))
columns$dist_lastused = abs(as.numeric(difftime(strptime(columns$lastused1, "%Y/%m/%d %H:%M:%S"),
                                      strptime(columns$lastused2, "%Y/%m/%d %H:%M:%S"))))
columns$dist_lastused[is.na(columns$dist_lastused)] = 0
columns$dist_size = apply(columns[,c('size1','size2')], 1, function(x) sd(x))

# Decisao. Sendo: 0 para diferente e 1 para similar
columns$decisao = 0
columns$decisao[columns$lv_dicTab == 0 & columns$cos_dicCol == 0] = 1
columns$decisao[columns$cos_dicTab == 0 & columns$cos_dicCol == 0] = 1
columns$decisao[columns$lv_dicTab == 0 & columns$lv_dicCol == 0] = 1

# Proporcao
table(columns$decisao)
```

```
##
##       0       1
## 9267457    1208
```

```r
prop.table(table(columns$decisao))
```

```
##
##            0            1
## 0.9998696684 0.0001303316
```

```r
a = columns[columns$decisao == 1,]

# Verifica os iguais
length(unique(a$id1))
```

```
## [1] 721
```

```r
length(unique(columns$id1))
```

```
## [1] 4305
```

```r
b = as.data.frame(table(a$dicCol1))
b = b[b$Freq > 2,]

# Salva
write.csv2(x = columns, file = "columns.csv", row.names = TRUE)
```

```r
# Carrega
library(readr)
columns = read_csv2("columns.csv")
columns = as.data.frame(columns[,2:50])
columns$id = paste(columns$server1, columns$schema1, columns$table1, sep = " ")
columns$idz = paste(columns$server2, columns$schema2, columns$table2, sep = " ")

# Agrega as colunas por tabela
a = columns[,c(50, 3)]
a = aggregate(server1 ~ id, a, FUN=unique)
b = columns[,c(51, 4)]
b = aggregate(server2 ~ idz, b, FUN=unique)
c = columns[,c(50, 5)]
c = aggregate(schema1 ~ id, c, FUN=unique)
d = columns[,c(51, 6)]
d = aggregate(schema2 ~ idz, d, FUN=unique)
e = columns[,c(50, 7)]
e = aggregate(table1 ~ id, e, FUN=unique)
f = columns[,c(51, 8)]
f = aggregate(table2 ~ idz, f, FUN=unique)
g = columns[,c(50, 13)]
g = aggregate(dicTab1 ~ id, g, FUN=unique)
h = columns[,c(51, 14)]
h = aggregate(dicTab2 ~ idz, h, FUN=unique)

i = columns[,c(50, 25)]
i = aggregate(lastused1 ~ id, i, FUN=unique)
j = columns[,c(51, 26)]
j = aggregate(lastused2 ~ idz, j, FUN=unique)

# Faz o merge de tudo e salva na nova variavel tables
tables = merge(a, b)
tables = merge(tables, c)
tables = merge(tables, d)
tables = merge(tables, e)
tables = merge(tables, f)
tables = merge(tables, g)
tables = merge(tables, h)
tables = merge(tables, i)
tables = merge(tables, j)
rm(a,b,c,d,e,f,g,h,i,j)

# Remove as comparacoes com ele mesmo
library(dplyr)
tables = tables %>% filter(id != idz)

# Insere os demais dados dentro do data.frame de agregacao de tabelas
tables$nrow1 = columns$nrow1[match(tables$id, columns$id)]
tables$nrow2 = columns$nrow2[match(tables$idz, columns$idz)]
tables$size1 = columns$size1[match(tables$id, columns$id)]
tables$size2 = columns$size2[match(tables$idz, columns$idz)]

tables$lv_schema = stringdist(tables$schema1, tables$schema2, method = "lv")
```

```r
tables$cos_schema = stringdist(tables$schema1, tables$schema2, method = "cosine")
tables$lv_tab = stringdist(tables$table1, tables$table2, method = "lv")
tables$cos_tab = stringdist(tables$table1, tables$table2, method = "cosine")
tables$lv_dicTab = stringdist(tables$dicTab1, tables$dicTab2, method = "lv")
tables$cos_dicTab = stringdist(tables$dicTab1, tables$dicTab2, method = "cosine")

# Calcula a qtd de colunas
a = aggregate(col1 ~ server1 + schema1 + table1, columns, FUN = length)
b = aggregate(col2 ~ server2 + schema2 + table2, columns, FUN = length)
tables = merge(tables, a)
tables = merge(tables, b)

# Calcula a distancia entre as variaveis numericas
tables$dist_nrow = apply(tables[,c('nrow1','nrow2')], 1, function(x) sd(x))
tables$dist_lastused = abs(as.numeric(difftime(strptime(tables$lastused1, "%Y/%m/%d %H:%M:%S"),
                                       strptime(tables$lastused2, "%Y/%m/%d %H:%M:%S"))))
tables$dist_lastused[is.na(tables$dist_lastused)] = 0
tables$dist_size = apply(tables[,c('size1','size2')], 1, function(x) sd(x))

# Decisao. Sendo: 0 para diferente e 1 para similar
tables$decisao = 0
tables$decisao[tables$lv_dicTab == 0] = 1
tables$decisao[tables$cos_dicTab == 0] = 1

# Se nao tiver, zera
tables$decisao[is.na(tables$decisao)] = 0

# Salva
write.csv2(x = tables, file = "tables.csv", row.names = TRUE)
```

```r
# Carrega
library(readr)
tables = read_csv2("tables.csv")
tables = as.data.frame(tables[,2:29])

# Agrega as tabelas por schema
tables$id_schema = paste(tables$server1, tables$schema1, sep = " ")
tables$id_schema2 = paste(tables$server2, tables$schema2, sep = " ")

# Agrega as tabelas por schema
a = tables[,c(29, 4)]
a = aggregate(server1 ~ id_schema, a, FUN=unique)
b = tables[,c(30, 1)]
b = aggregate(server2 ~ id_schema2, b, FUN=unique)
c = tables[,c(29, 5)]
c = aggregate(schema1 ~ id_schema, c, FUN=unique)
d = tables[,c(30, 2)]
d = aggregate(schema2 ~ id_schema2, d, FUN=unique)

# Faz o merge de tudo e salva na nova variavel
schemas = merge(a, b)
schemas = merge(schemas, c)
schemas = merge(schemas, d)
rm(a,b,c,d)
```

```r
# Remove as comparacoes com ele mesmo
library(dplyr)
schemas = schemas %>% filter(id_schema != id_schema2)

# Carrega a qtd de tabelas de cada schema
a = aggregate(table1 ~ server1 + schema1, tables, FUN = length)
b = aggregate(table2 ~ server2 + schema2, tables, FUN = length)
schemas = merge(schemas, a)
schemas = merge(schemas, b)

# Salva
write.csv2(x = schemas, file = "schemas.csv", row.names = TRUE)
```

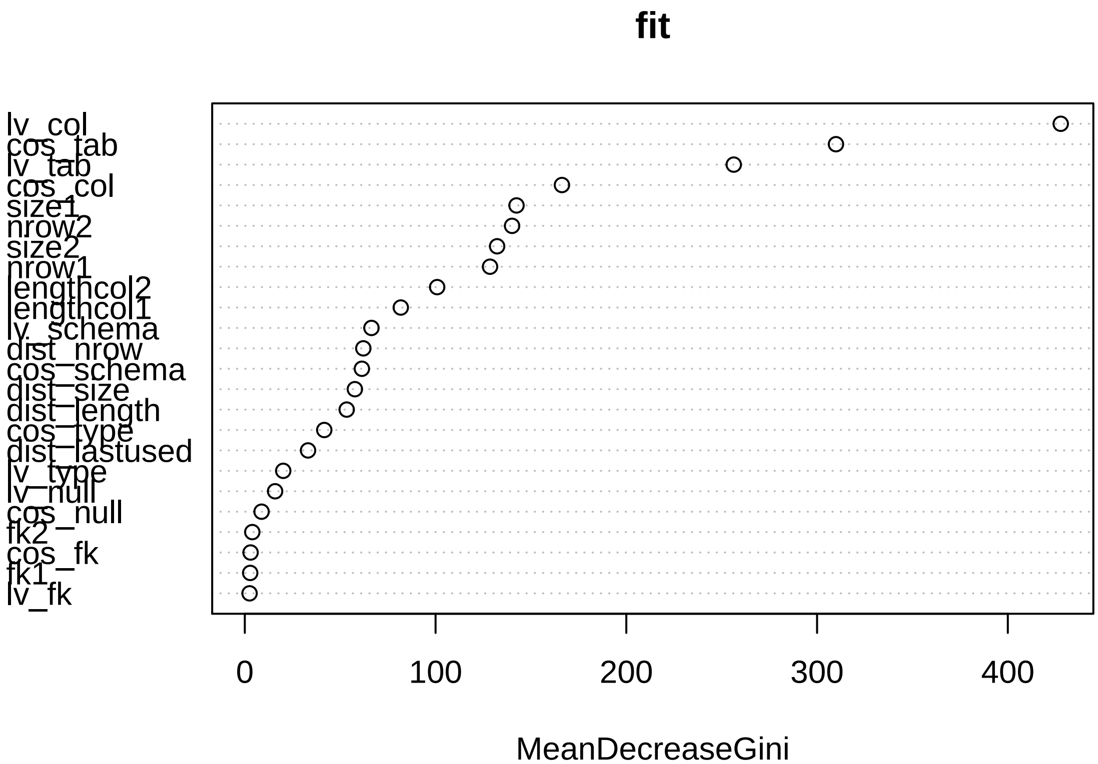
```r
# Carrega os dados
library(readr)
columns = read_csv2("columns.csv")

# Mantém apenas as variaveis numéricas e retira as variaveis do dicionario de dados: 24 variaveis
columns = as.data.frame(columns[,c(18:21,24,25,28:37, 42:50)])

# Verifica se tem NA
anyNA(columns)
```

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

```r
# Remove todos os objetos, menos columns
rm(list=setdiff(ls(), "columns"))
gc()
```

```
##              used   (Mb) gc trigger   (Mb)   max used    (Mb)
## Ncells    580737   31.1   11409346  609.4   43523207  2324.4
## Vcells 166110427 1267.4 1280676678 9770.8 3256066712 24841.9
```

```r
# Descobre as variaveis mais importantes
library(randomForest)
fit = randomForest(formula = as.factor(decisao) ~ .,
                   data = columns, ntree = 50)
varImpPlot(fit)
```




```r
# Divide em treino, validacao e teste
set.seed(0)
amostra_col = sample(3, nrow(columns), replace = TRUE, prob = c(.6,.2,.2))
treino_col = columns[amostra_col == 1,]
validacao_col = columns[amostra_col == 2,]
teste_col = columns[amostra_col == 3,]
table(treino_col$decisao)
```

```
##
##       0       1
## 5562795     711
```

```r
table(validacao_col$decisao)
```

```
##
##       0       1
## 1851888     245
```

```r
table(teste_col$decisao)
```

```
##
##       0       1
## 1852774     252
```

```r
# Exporta dados processados para arquivo .csv
write.csv2(x = treino_col, file = "treino_col.csv", row.names = TRUE)
write.csv2(x = validacao_col, file = "validacao_col.csv", row.names = TRUE)
write.csv2(x = teste_col, file = "teste_col.csv", row.names = TRUE)

# Remove todos os objetos e faz garbage collector
rm(list = ls())
gc()
```

```
##              used   (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells   594916   31.8   19098510  1020.0   43523207   2324.4
## Vcells 17834271 136.1 4577505300 34923.6 5718066062 43625.4
```

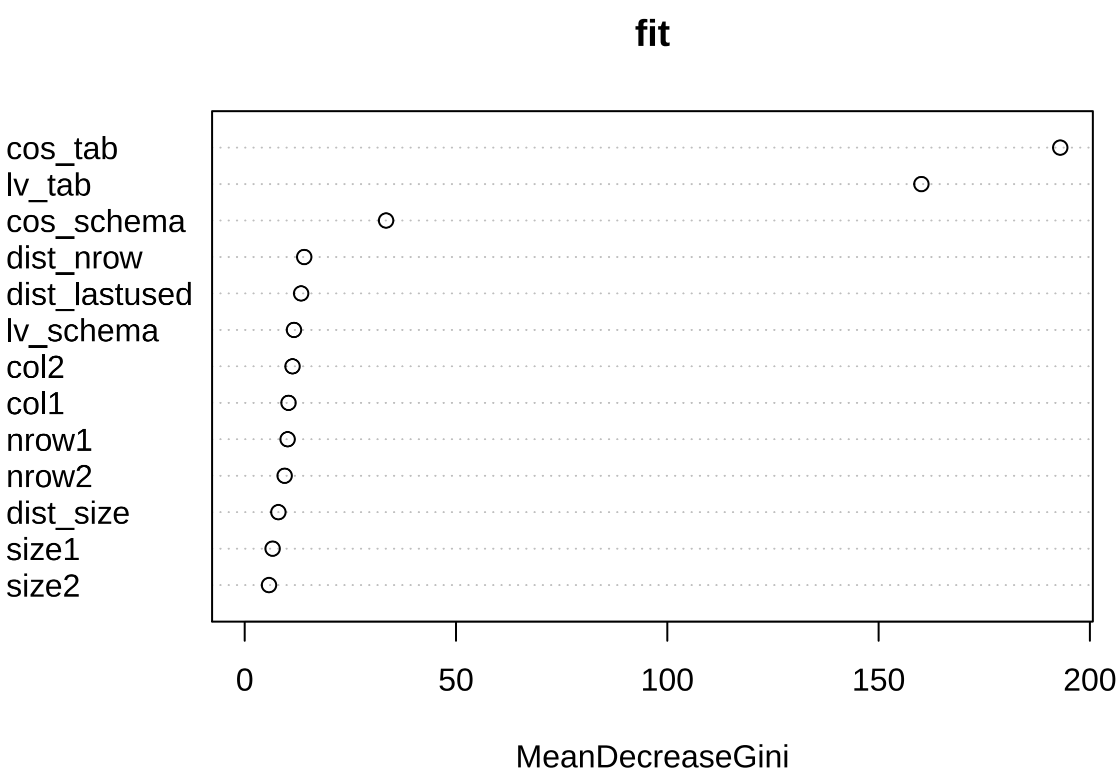
```r
# Carrega os dados
library(readr)
tables = read_csv2("tables.csv")

# Mantém apenas as variaveis numéricas e retira as variaveis do dicionario de dados: 24 variaveis
tables = as.data.frame(tables[,c(14:21,24:29)])

# Verifica se tem NA
anyNA(tables)
```

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

```r
# Remove todos os objetos, menos tables
rm(list=setdiff(ls(), "tables"))
gc()
```

```
##              used   (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells   596465   31.9   15278808   816.0   43523207   2324.4
## Vcells 20947776 159.9 3662004240 27938.9 5718066062 43625.4
```

```r
# Descobre as variaveis mais importantes
library(randomForest)
fit = randomForest(formula = as.factor(decisao) ~ .,
                     data = tables, ntree = 100)
varImpPlot(fit)
```




```r
# Divide em treino, validacao e teste
set.seed(0)
```

```r
amostra_tab = sample(3, nrow(tables), replace = TRUE, prob = c(.6,.2,.2))
treino_tab = tables[amostra_tab == 1,]
validacao_tab = tables[amostra_tab == 2,]
teste_tab = tables[amostra_tab == 3,]
table(treino_tab$decisao)
```

```
##
##      0      1
## 207576    138
```

```r
table(validacao_tab$decisao)
```

```
##
##     0     1
## 69059    50
```

```r
table(teste_tab$decisao)
```

```
##
##     0     1
## 69453    56
```

```r
# Exporta dados processados para arquivo .csv
write.csv2(x = treino_tab, file = "treino_tab.csv", row.names = TRUE)
write.csv2(x = validacao_tab, file = "validacao_tab.csv", row.names = TRUE)
write.csv2(x = teste_tab, file = "teste_tab.csv", row.names = TRUE)

# Remove todos os objetos e faz garbage collector
rm(list = ls())
gc()
```

```
##            used  (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells   598730  32.0   12223046   652.8   43523207  2324.4
## Vcells 17849568 136.2 2929603392 22351.2 5718066062 43625.4
```

Data Mining Sequencial - com 1 core

1. GLM
2. Random Forest - RF
3. GBM

```r
# Carrega os dados se estiverem em diretório local
library(readr)
# Colunas
treino_col = read.csv2("treino_col.csv", sep = ";")
validacao_col = read.csv2("validacao_col.csv", sep = ";")
teste_col = read.csv2("teste_col.csv", sep = ";")
treino_col = as.data.frame(treino_col[,2:26])
validacao_col = as.data.frame(validacao_col[,2:26])
teste_col = as.data.frame(teste_col[,2:26])

# Tabelas
treino_tab = read.csv2("treino_tab.csv", sep = ";")
validacao_tab = read.csv2("validacao_tab.csv", sep = ";")
teste_tab = read.csv2("teste_tab.csv", sep = ";")
```

```r
treino_tab = as.data.frame(treino_tab[,2:15])
validacao_tab = as.data.frame(validacao_tab[,2:15])
teste_tab = as.data.frame(teste_tab[,2:15])
```

```r
library(MASS)
library(mlbench)
library(caret)

################################
# 1. GLM
# Inicia o contador do tempo de execução
start.time <- Sys.time()

#####
# Logistic Regression - GLM - Usando 10 folds cross validation e repetindo 3 vezes - COLUMN
set.seed(10)
glm_fit_seq_col <- train(as.factor(decisao) ~ ., data = treino_col, method="glm",
                  trControl=trainControl(method="repeatedcv", number=10, repeats=3, sampling = "down"),
                  family = binomial("logit"), maxit = 10)

# Predictions
glm_pred_seq_col <- predict(glm_fit_seq_col, newdata=teste_col)
cf_glm_seq_col = as.matrix(confusionMatrix(glm_pred_seq_col, teste_col$decisao))

# Precisao tp/(tp+fp)
precisao_glm_seq_col = cf_glm_seq_col[1,1]/sum(cf_glm_seq_col[1,1:2])
precisao_glm_seq_col
```

```
## [1] 1
```

```r
# Recall: tp/(tp + fn)
recall_glm_seq_col = cf_glm_seq_col [1,1]/sum(cf_glm_seq_col [1:2,1])
recall_glm_seq_col
```

```
## [1] 0.9929155
```

```r
# F-Score: 2 * precision * recall /(precision + recall)
fmeasure_glm_seq_col = 2 * precisao_glm_seq_col * recall_glm_seq_col / (precisao_glm_seq_col + recall_g
fmeasure_glm_seq_col
```

```
## [1] 0.9964452
```

```r
###
# TABLES
set.seed(10)
glm_fit_seq_tab <- train(as.factor(decisao) ~ ., data = treino_tab, method="glm",
                  trControl=trainControl(method="repeatedcv", number=10, repeats=3, sampling = "down"),
                  family = binomial("logit"), maxit = 10)

# Predictions
pred_glm_seq_tab <- predict(glm_fit_seq_tab, newdata=teste_tab)
cf_glm_seq_tab = as.matrix(confusionMatrix(pred_glm_seq_tab, teste_tab$decisao))

# Precisao tp/(tp+fp)
precisao_glm_seq_tab = cf_glm_seq_tab[1,1]/sum(cf_glm_seq_tab[1,1:2])
precisao_glm_seq_tab
```

```
## [1] 0.999897
```

```r
# Recall: tp/(tp + fn)
recall_glm_seq_tab = cf_glm_seq_tab[1,1]/sum(cf_glm_seq_tab[1:2,1])
recall_glm_seq_tab
```

```
## [1] 0.9779563
```

```r
# F-Score: 2 * precision * recall /(precision + recall)
fmeasure_glm_seq_tab = 2 * precisao_glm_seq_tab * recall_glm_seq_tab / (precisao_glm_seq_tab + recall_gl
fmeasure_glm_seq_tab
```

```
## [1] 0.9888049
```

```r
# Conta o tempo de execução - GLM
end.time <- Sys.time()
time.taken_glm_trad <- end.time - start.time
time.taken_glm_trad
```

```
## Time difference of 12.03564 mins
```

```r
#########################
# 2. RANDOM FOREST
library(randomForest)
library(caret)

# Inicia o contador do tempo de execução
start.time <- Sys.time()

### COLUMN
# Random Forest - com cross validation - Usando 10 folds cross validation e repetindo 3 vezes
set.seed(10)
rf_fit_seq_col <- train(as.factor(decisao) ~ ., data = treino_col,  method="rf",
                 trControl=trainControl(method="repeatedcv", number=10, repeats=3, sampling = "down"))

# Predictions
rf_pred_seq_col <- predict(rf_fit_seq_col, newdata=teste_col)
cf_rf_seq_col = as.matrix(confusionMatrix(rf_pred_seq_col, teste_col$decisao))

# Precisao tp/(tp+fp)
precisao_rf_seq_col = cf_rf_seq_col[1,1]/sum(cf_rf_seq_col[1,1:2])
precisao_rf_seq_col
```

```
## [1] 1
```

```r
# Recall: tp/(tp + fn)
recall_rf_seq_col = cf_rf_seq_col[1,1]/sum(cf_rf_seq_col[1:2,1])
recall_rf_seq_col
```

```
## [1] 0.9971675
```

```r
# F-Score: 2 * precision * recall /(precision + recall)
fmeasure_rf_seq_col= 2 * precisao_rf_seq_col * recall_rf_seq_col / (precisao_rf_seq_col + recall_rf_seq_
fmeasure_rf_seq_col
```

```
## [1] 0.9985817
```

```r
### TABLE
# Random Forest - com cross validation - Usando 10 folds cross validation e repetindo 3 vezes
```

```r
set.seed(10)
rf_fit_seq_tab <- train(as.factor(decisao) ~ ., data = treino_tab,  method="rf",
                 trControl=trainControl(method="repeatedcv", number=10, repeats=3, sampling = "down"))

# Predictions
rf_pred_seq_tab <- predict(rf_fit_seq_tab, newdata=teste_tab)
cf_rf_seq_tab = as.matrix(confusionMatrix(rf_pred_seq_tab, teste_tab$decisao))

# Precisao tp/(tp+fp)
precisao_rf_seq_tab = cf_rf_seq_tab [1,1]/sum(cf_rf_seq_tab [1,1:2])
precisao_rf_seq_tab
```

```
## [1] 1
```

```r
# Recall: tp/(tp + fn)
recall_rf_seq_tab = cf_rf_seq_tab [1,1]/sum(cf_rf_seq_tab [1:2,1])
recall_rf_seq_tab
```

```
## [1] 0.998445
```

```r
# F-Score: 2 * precision * recall /(precision + recall)
fmeasure_rf_seq_tab = 2 * precisao_rf_seq_tab * recall_rf_seq_tab / (precisao_rf_seq_tab + recall_rf_se
fmeasure_rf_seq_tab
```

```
## [1] 0.9992219
```

```r
# Conta o tempo de execução - RF
end.time <- Sys.time()
time.taken_rf_trad <- end.time - start.time
time.taken_rf_trad
```

```
## Time difference of 48.2618 mins
```

```r
##########################
# 3. GBM - gradient boost machines:
library(gbm)

# Inicia o contador do tempo de execução
start.time <- Sys.time()

#### COLUMN
# GBM - com cross validation - Usando 10 folds cross validation e repetindo 3 vezes
set.seed(10)
gbm_fit_seq_col <- train(as.factor(decisao) ~ ., data = treino_col,  method="gbm",
                 trControl=trainControl(method="repeatedcv", number=10, repeats=3, sampling = "down"),
                 verbose = FALSE)

# Predictions
gbm_pred_seq_col <- predict(gbm_fit_seq_col, newdata=teste_col, n.trees = 10)
cf_gbm_seq_col = as.matrix(confusionMatrix(gbm_pred_seq_col, teste_col$decisao))

# Precisao tp/(tp+fp)
precisao_gbm_seq_col = cf_gbm_seq_col[1,1]/sum(cf_gbm_seq_col[1,1:2])
precisao_gbm_seq_col
```

```
## [1] 1
```

```r
# Recall tp/(tp + fn)
recall_gbm_seq_col = cf_gbm_seq_col[1,1]/sum(cf_gbm_seq_col[1:2,1])
recall_gbm_seq_col
```

```
## [1] 0.9952396
```

```r
# F-Score: 2 * precision * recall /(precision + recall)
fmeasure_gbm_seq_col = 2 * precisao_gbm_seq_col * recall_gbm_seq_col / (precisao_gbm_seq_col + recall_g
fmeasure_gbm_seq_col
```

```
## [1] 0.9976141
```

```r
#### TABLE
# GBM - com cross validation - Usando 10 folds cross validation e repetindo 3 vezes
set.seed(10)
gbm_fit_seq_tab <- train(as.factor(decisao) ~ ., data = treino_tab,  method="gbm",
                 trControl=trainControl(method="repeatedcv", number=10, repeats=3, sampling = "down"),
                 verbose = FALSE)

# Predictions
pred_gbm_seq_tab <- predict(gbm_fit_seq_tab, newdata=teste_tab, n.trees = 10)
cf_gbm_seq_tab = as.matrix(confusionMatrix(pred_gbm_seq_tab, teste_tab$decisao))

# Precisao tp/(tp+fp)
precisao_gbm_seq_tab = cf_gbm_seq_tab[1,1]/sum(cf_gbm_seq_tab[1,1:2])
precisao_gbm_seq_tab
```

```
## [1] 1
```

```r
# Recall tp/(tp + fn)
recall_gbm_seq_tab = cf_gbm_seq_tab[1,1]/sum(cf_gbm_seq_tab[1:2,1])
recall_gbm_seq_tab
```

```
## [1] 0.9959397
```

```r
# F-Score: 2 * precision * recall /(precision + recall)
fmeasure_gbm_seq_tab = 2 * precisao_gbm_seq_tab * recall_gbm_seq_tab/ (precisao_gbm_seq_tab + recall_gbm
fmeasure_gbm_seq_tab
```

```
## [1] 0.9979657
```

```r
# Conta o tempo de execução - GBM
end.time <- Sys.time()
time.taken_gbm_trad <- end.time - start.time
time.taken_gbm_trad
```

```
## Time difference of 40.36751 mins
```

```r
# Remove todos os objetos e faz garbage collector
rm(list = ls())
gc()
```

```
##             used  (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells  1739445  92.9   17013618   908.7   43523207  2324.4
## Vcells 18712969 142.8 1728091189 13184.3 5718066062 43625.4
```

Mineração de Dados - Processamento Paralelo

1. GLM
2. Random Forest - RF
3. GBM

```
# Inicia o contador do tempo de execução
start.time <- Sys.time()

# Identifica quantos cores
library(doParallel)
library(doMC)
detectCores()
```

```
## [1] 8
```

```
nr_cores <- 7
registerDoMC(nr_cores)
getDoParWorkers()
```

```
## [1] 7
```

```
# Inicia o H2O
library(h2o)
h2o.init(max_mem_size = "30G", enable_assertions = FALSE)
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     /tmp/RtmpvTN044/h2o_rstudio_started_from_r.out
##     /tmp/RtmpvTN044/h2o_rstudio_started_from_r.err
##
##
## Starting H2O JVM and connecting: ... Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         2 seconds 661 milliseconds
##     H2O cluster version:        3.10.5.3
##     H2O cluster version age:    18 days
##     H2O cluster name:           H2O_started_from_R_rstudio_ozg324
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   26.67 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 3.4.0 (2017-04-21)
```

```
# Carregar dados locais no H2O - COLUMN
pathToData <- h2o:::.h2o.locate("treino_col.csv")
treino_col <- h2o.importFile(pathToData,  header = T)
```

```
##
```

```
  |
  |                                                                 |   0%
  |
  |================                                                 |  25%
  |
  |====================                                             |  31%
  |
  |================================                                 |  50%
  |
  |===========================================                      |  66%
  |
  |=====================================================            |  81%
  |
  |=================================================================| 100%
```

```r
pathToData <- h2o:::.h2o.locate("validacao_col.csv")
validacao_col <- h2o.importFile(pathToData,  header = T)
```

```
##
  |
  |                                                                 |   0%
  |
  |================                                                 |  25%
  |
  |=================================================================| 100%
```

```r
pathToData <- h2o:::.h2o.locate("teste_col.csv")
teste_col <- h2o.importFile(pathToData,  header = T)
```

```
##
  |
  |                                                                 |   0%
  |
  |====================                                             |  31%
  |
  |=================================================================| 100%
```

```r
# Carregar dados locais no H2O - TABLE
pathToData <- h2o:::.h2o.locate("treino_tab.csv")
treino_tab <- h2o.importFile(pathToData,  header = T)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```r
pathToData <- h2o:::.h2o.locate("validacao_tab.csv")
validacao_tab <- h2o.importFile(pathToData,  header = T)
```

```
##
  |
  |                                                                 |   0%
  |
  |====                                                             |   6%
  |
  |=================================================================| 100%
```

```r
pathToData <- h2o:::.h2o.locate("teste_tab.csv")
teste_tab <- h2o.importFile(pathToData,  header = T)
```

```
##
  |
  |                                                                  |   0%
  |
  |==============                                                    |  22%
  |
  |==================================================================| 100%
```

```r
# Transforma a classe em factor
teste_col$classe <- as.factor(teste_col$decisao)
validacao_col$classe <- as.factor(validacao_col$decisao)
treino_col$classe <- as.factor(treino_col$decisao)
h2o.levels(treino_col$classe)
```

```
## [1] "0" "1"
```

```r
teste_tab$classe <- as.factor(teste_tab$decisao)
validacao_tab$classe <- as.factor(validacao_tab$decisao)
treino_tab$classe <- as.factor(treino_tab$decisao)
h2o.levels(treino_tab$classe)
```

```
## [1] "0" "1"
```

```r
# Conta o tempo de execução - Pre-processamento
end.time <- Sys.time()
time.taken <- end.time - start.time
time.taken
```

```
## Time difference of 30.85532 secs
```

```r
################
# 1. GLM
# Inicia o contador do tempo de execução
start.time <- Sys.time()

### COLUMN
# Modelo GLM com validacao
glm_fit_par_col <- h2o.glm(y = "classe",
                    training_frame = treino_col,
                    validation_frame = validacao_col,
                    family = "binomial",
                    balance_classes = TRUE,
                    seed = 1,
                    lambda_search = TRUE)
```

```
##
  |
  |                                                                  |   0%
  |
  |=                                                                 |   1%
  |
  |=                                                                 |   2%
  |
  |==                                                                |   3%
```

```
|
|===                                                              |   4%
|
|===                                                              |   5%
|
|====                                                             |   6%
|
|=====                                                            |   7%
|
|=====                                                            |   8%
|
|======                                                           |   9%
|
|======                                                           |  10%
|
|=======                                                          |  11%
|
|========                                                         |  12%
|
|========                                                         |  13%
|
|=========                                                        |  14%
|
|==========                                                       |  15%
|
|==========                                                       |  16%
|
|===========                                                      |  17%
|
|============                                                     |  18%
|
|============                                                     |  19%
|
|=============                                                    |  20%
|
|==============                                                   |  21%
|
|==============                                                   |  22%
|
|===============                                                  |  23%
|
|================                                                 |  24%
|
|================                                                 |  25%
|
|=================                                                |  26%
|
|==================                                               |  27%
|
|==================                                               |  28%
|
|===================                                              |  29%
|
|====================                                             |  30%
```

```
|
|====================                                             |  31%
|
|=====================                                            |  32%
|
|=====================                                            |  33%
|
|======================                                           |  34%
|
|=======================                                          |  35%
|
|=======================                                          |  36%
|
|========================                                         |  37%
|
|=========================                                        |  38%
|
|=========================                                        |  39%
|
|==========================                                       |  40%
|
|===========================                                      |  41%
|
|===========================                                      |  42%
|
|============================                                     |  43%
|
|=============================                                    |  44%
|
|=============================                                    |  45%
|
|==============================                                   |  46%
|
|===============================                                  |  47%
|
|===============================                                  |  48%
|
|================================                                 |  49%
|
|================================                                 |  50%
|
|=================================                                |  51%
|
|==================================                               |  52%
|
|==================================                               |  53%
|
|===================================                              |  54%
|
|====================================                             |  55%
|
|====================================                             |  56%
|
|=====================================                            |  57%
```

```
|
|=====================================                            |  58%
|
|=====================================                            |  59%
|
|======================================                           |  60%
|
|=======================================                          |  61%
|
|=======================================                          |  62%
|
|========================================                         |  63%
|
|=========================================                        |  64%
|
|=========================================                        |  65%
|
|==========================================                       |  66%
|
|===========================================                      |  67%
|
|===========================================                      |  68%
|
|============================================                     |  69%
|
|=============================================                    |  70%
|
|=============================================                    |  71%
|
|==============================================                   |  72%
|
|==============================================                   |  73%
|
|===============================================                  |  74%
|
|================================================                 |  75%
|
|================================================                 |  76%
|
|=================================================                |  77%
|
|==================================================               |  78%
|
|==================================================               |  79%
|
|===================================================              |  80%
|
|====================================================             |  81%
|
|====================================================             |  82%
|
|=====================================================            |  83%
|
|======================================================           |  84%
```

```
  |
  |=======================================================          |  85%
  |
  |========================================================         |  86%
  |
  |=========================================================        |  87%
  |
  |=========================================================        |  88%
  |
  |==========================================================       |  89%
  |
  |==========================================================       |  90%
  |
  |===========================================================      |  91%
  |
  |============================================================     |  92%
  |
  |============================================================     |  93%
  |
  |=============================================================    |  94%
  |
  |==============================================================   |  95%
  |
  |==============================================================   |  96%
  |
  |===============================================================  |  97%
  |
  |================================================================ |  98%
  |
  |================================================================ |  99%
  |
  |=================================================================| 100%
```

```r
# save the model
#h2o.saveModel(object= glm_fit2, path=getwd(), force=TRUE)
# compare the performance of the two GLMs.
glm_perf_par_col <- h2o.performance(model = glm_fit_par_col, newdata = teste_col)

# Print the model performance
glm_perf_par_col
```

```
## H2OBinomialMetrics: glm
##
## MSE:  2.298233e-12
## RMSE:  1.515992e-06
## LogLoss:  4.059898e-08
## Mean Per-Class Error:  0
## AUC:  1
## Gini:  1
## R^2:  1
## Residual Deviance:  0.1504619
## AIC:  4.150462
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##              0   1    Error        Rate
```

```
## 0      1852774   0 0.000000  =0/1852774
## 1            0 252 0.000000      =0/252
## Totals 1852774 252 0.000000  =0/1853026
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.999870 1.000000   0
## 2                       max f2  0.999870 1.000000   0
## 3                 max f0point5  0.999870 1.000000   0
## 4                 max accuracy  0.999870 1.000000   0
## 5                max precision  0.999870 1.000000   0
## 6                   max recall  0.999870 1.000000   0
## 7              max specificity  0.999870 1.000000   0
## 8             max absolute_mcc  0.999870 1.000000   0
## 9   max min_per_class_accuracy  0.999870 1.000000   0
## 10 max mean_per_class_accuracy  0.999870 1.000000   0
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F
```

```r
h2o.auc(glm_perf_par_col)
```

```
## [1] 1
```

```r
h2o.F1(glm_perf_par_col)
```

```
##      threshold           f1
## 1 9.998700e-01 1.0000000000
## 2 2.292402e-08 0.0002719506
```

```r
h2o.precision(glm_perf_par_col)
```

```
##      threshold    precision
## 1 9.998700e-01 1.0000000000
## 2 2.292402e-08 0.0001359938
```

```r
h2o.recall(glm_perf_par_col)
```

```
##      threshold tpr
## 1 9.998700e-01   1
## 2 2.292402e-08   1
```

```r
### TABLE
# Modelo GLM com validacao
glm_fit_par_tab <- h2o.glm(y = "classe",
                    training_frame = treino_tab,
                    validation_frame = validacao_tab,
                    family = "binomial",
                    balance_classes = TRUE,
                    seed = 1,
                    lambda_search = TRUE)
```

```
##
  |
  |                                                                  |   0%
  |
  |========                                                          |  13%
  |
  |===================                                               |  29%
```

```
  |
  |=============================                                        |  44%
  |
  |=======================================                              |  60%
  |
  |=================================================                    |  76%
  |
  |===========================================================          |  91%
  |
  |=====================================================================| 100%
```

```r
# save the model
#h2o.saveModel(object= glm_fit2, path=getwd(), force=TRUE)
# compare the performance of the two GLMs.
glm_perf_par_tab <- h2o.performance(model = glm_fit_par_tab, newdata = teste_tab)

# Print the model performance
glm_perf_par_tab
```

```
## H2OBinomialMetrics: glm
##
## MSE:  1.595827e-07
## RMSE:  0.000399478
## LogLoss:  2.076279e-05
## Mean Per-Class Error:  0
## AUC:  1
## Gini:  1
## R^2:  0.9998018
## Residual Deviance:  2.886402
## AIC:  6.886402
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##            0  1    Error      Rate
## 0      69453  0 0.000000  =0/69453
## 1          0 56 0.000000     =0/56
## Totals 69453 56 0.000000  =0/69509
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.985930 1.000000   0
## 2                       max f2  0.985930 1.000000   0
## 3                 max f0point5  0.985930 1.000000   0
## 4                 max accuracy  0.985930 1.000000   0
## 5                max precision  0.985930 1.000000   0
## 6                   max recall  0.985930 1.000000   0
## 7              max specificity  0.985930 1.000000   0
## 8             max absolute_mcc  0.985930 1.000000   0
## 9   max min_per_class_accuracy  0.985930 1.000000   0
## 10 max mean_per_class_accuracy  0.985930 1.000000   0
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F
```

```r
h2o.auc(glm_perf_par_tab)
```

```
## [1] 1
```

```r
h2o.F1(glm_perf_par_tab)
```

```
##      threshold          f1
## 1 9.859298e-01 1.000000000
## 2 9.354107e-06 0.001610005
```

```r
h2o.precision(glm_perf_par_tab)
```

```
##      threshold    precision
## 1 9.859298e-01 1.0000000000
## 2 9.354107e-06 0.0008056511
```

```r
h2o.recall(glm_perf_par_tab)
```

```
##      threshold tpr
## 1 9.859298e-01   1
## 2 9.354107e-06   1
```

```r
# Conta o tempo de execução - GLM
end.time <- Sys.time()
time.taken_glm_paral <- end.time - start.time
time.taken_glm_paral
```

```
## Time difference of 3.405605 mins
```

```r
###################
# 2. Random Forest - RF
# Inicia o contador do tempo de execução
start.time <- Sys.time()

## TABLE
rf_fit_par_col <- h2o.randomForest(y = "classe",
                            training_frame = treino_col,
                            model_id = "rf_fit_par_col",
                            validation_frame = validacao_col,
                            balance_classes = TRUE,
                            ntrees = 50,
                            seed = 1)
```

```
##
  |
  |                                                                      |   0%
  |
  |=                                                                     |   2%
  |
  |===                                                                   |   4%
  |
  |====                                                                  |   6%
  |
  |=====                                                                 |   8%
  |
  |======                                                                |  10%
  |
  |========                                                              |  12%
  |
  |=========                                                             |  14%
  |
```

```
|==========                                                       |  16%
|
|============                                                     |  18%
|
|=============                                                    |  20%
|
|==============                                                   |  22%
|
|================                                                 |  24%
|
|=================                                                |  26%
|
|==================                                               |  28%
|
|====================                                             |  30%
|
|=====================                                            |  32%
|
|======================                                           |  34%
|
|=======================                                          |  36%
|
|=========================                                        |  38%
|
|==========================                                       |  40%
|
|===========================                                      |  42%
|
|=============================                                    |  44%
|
|==============================                                   |  46%
|
|===============================                                  |  48%
|
|================================                                 |  50%
|
|==================================                               |  52%
|
|===================================                              |  54%
|
|====================================                             |  56%
|
|======================================                           |  58%
|
|=======================================                          |  60%
|
|========================================                         |  62%
|
|==========================================                       |  64%
|
|===========================================                      |  66%
|
|============================================                     |  68%
|
```

```
  |==============================================                   |  70%
  |
  |===============================================                  |  72%
  |
  |================================================                 |  74%
  |
  |=================================================                |  76%
  |
  |===================================================              |  78%
  |
  |====================================================             |  80%
  |
  |=====================================================            |  82%
  |
  |=======================================================          |  84%
  |
  |========================================================         |  86%
  |
  |=========================================================        |  88%
  |
  |==========================================================       |  90%
  |
  |============================================================     |  92%
  |
  |=============================================================    |  94%
  |
  |==============================================================   |  96%
  |
  |================================================================ |  98%
  |
  |=================================================================| 100%
```

```r
# save the model
#h2o.saveModel(object= rf_fit2, path=getwd(), force=TRUE)

# Compare the performance
rf_perf_par_col <- h2o.performance(model = rf_fit_par_col, newdata = teste_col)

# print the performance model
rf_perf_par_col
```

```
## H2OBinomialMetrics: drf
##
## MSE:  0.0001342768
## RMSE:  0.01158779
## LogLoss:  0.0006994244
## Mean Per-Class Error:  0
## AUC:  1
## Gini:  1
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##              0   1    Error        Rate
## 0      1852774   0 0.000000  =0/1852774
## 1            0 252 0.000000      =0/252
## Totals 1852774 252 0.000000  =0/1853026
```

```
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.000465 1.000000  22
## 2                       max f2  0.000465 1.000000  22
## 3                 max f0point5  0.000465 1.000000  22
## 4                 max accuracy  0.000465 1.000000  22
## 5                max precision  0.009767 1.000000   0
## 6                   max recall  0.000465 1.000000  22
## 7              max specificity  0.009767 1.000000   0
## 8             max absolute_mcc  0.000465 1.000000  22
## 9   max min_per_class_accuracy  0.000465 1.000000  22
## 10 max mean_per_class_accuracy  0.000465 1.000000  22
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F
```

```
h2o.auc(rf_perf_par_col)
```

```
## [1] 1
```

```
h2o.F1(rf_perf_par_col)
```

```
##     threshold          f1
## 1 0.009766810 0.007905138
## 2 0.009752272 0.038910506
## 3 0.007590136 0.090909091
## 4 0.007589997 0.429906542
## 5 0.007528265 0.434782609
##
## ---
##        threshold           f1
## 108 2.608437e-06 0.0013546239
## 109 2.298514e-06 0.0013534816
## 110 1.832039e-06 0.0013533217
## 111 1.030745e-06 0.0013532308
## 112 5.732990e-07 0.0013532199
## 113 0.000000e+00 0.0002719506
```

```
h2o.precision(rf_perf_par_col)
```

```
##     threshold precision
## 1 0.009766810         1
## 2 0.009752272         1
## 3 0.007590136         1
## 4 0.007589997         1
## 5 0.007528265         1
##
## ---
##        threshold    precision
## 108 2.608437e-06 0.0006777710
## 109 2.298514e-06 0.0006771991
## 110 1.832039e-06 0.0006771190
## 111 1.030745e-06 0.0006770735
## 112 5.732990e-07 0.0006770681
## 113 0.000000e+00 0.0001359938
```

```r
h2o.recall(rf_perf_par_col)
```

```
##     threshold         tpr
## 1 0.009766810 0.003968254
## 2 0.009752272 0.019841270
## 3 0.007590136 0.047619048
## 4 0.007589997 0.273809524
## 5 0.007528265 0.277777778
##
## ---
##        threshold tpr
## 108 2.608437e-06   1
## 109 2.298514e-06   1
## 110 1.832039e-06   1
## 111 1.030745e-06   1
## 112 5.732990e-07   1
## 113 0.000000e+00   1
```

```r
## COLUMN
rf_fit_par_tab <- h2o.randomForest(y = "classe",
                            training_frame = treino_tab,
                            model_id = "rf_fit_par_tab",
                            validation_frame = validacao_tab,
                            balance_classes = TRUE,
                            ntrees = 50,
                            seed = 1)
```

```
##
  |
  |                                                                      |   0%
  |
  |====                                                                  |   6%
  |
  |======                                                                |  10%
  |
  |=========                                                             |  14%
  |
  |============                                                          |  18%
  |
  |=========================                                             |  38%
  |
  |============================================                          |  68%
  |
  |======================================================================| 100%
```

```r
# save the model
#h2o.saveModel(object= rf_fit2, path=getwd(), force=TRUE)

# Compare the performance
rf_perf_par_tab <- h2o.performance(model = rf_fit_par_tab, newdata = teste_tab)

# print the performance model
rf_perf_par_tab
```

```
## H2OBinomialMetrics: drf
```

```
##
## MSE:  0.0003122894
## RMSE:  0.01767171
## LogLoss:  0.001335059
## Mean Per-Class Error:  0
## AUC:  1
## Gini:  1
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##            0  1    Error      Rate
## 0      69453  0 0.000000  =0/69453
## 1          0 56 0.000000     =0/56
## Totals 69453 56 0.000000  =0/69509
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.001549 1.000000  25
## 2                       max f2  0.001549 1.000000  25
## 3                 max f0point5  0.001549 1.000000  25
## 4                 max accuracy  0.001549 1.000000  25
## 5                max precision  1.000000 1.000000   0
## 6                   max recall  0.001549 1.000000  25
## 7              max specificity  1.000000 1.000000   0
## 8             max absolute_mcc  0.001549 1.000000  25
## 9   max min_per_class_accuracy  0.001549 1.000000  25
## 10 max mean_per_class_accuracy  0.001549 1.000000  25
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

```r
h2o.auc(rf_perf_par_tab)
```

```
## [1] 1
```

```r
h2o.F1(rf_perf_par_tab)
```

```
##   threshold        f1
## 1 1.0000000 0.1935484
## 2 0.9715527 0.3030303
## 3 0.9582799 0.4864865
## 4 0.9525862 0.5066667
## 5 0.9041960 0.5263158
##
## ---
##        threshold          f1
## 96  1.355278e-05 0.097816594
## 97  1.306356e-05 0.095400341
## 98  1.288816e-05 0.049689441
## 99  4.371464e-06 0.036234228
## 100 4.275271e-06 0.028105395
## 101 0.000000e+00 0.001610005
```

```r
h2o.precision(rf_perf_par_tab)
```

```
##   threshold precision
## 1 1.0000000         1
## 2 0.9715527         1
```

```
## 3 0.9582799         1
## 4 0.9525862         1
## 5 0.9041960         1
##
## ---
##         threshold    precision
## 96  1.355278e-05 0.0514233242
## 97  1.306356e-05 0.0500894454
## 98  1.288816e-05 0.0254777070
## 99  4.371464e-06 0.0184514003
## 100 4.275271e-06 0.0142529906
## 101 0.000000e+00 0.0008056511
```

```
h2o.recall(rf_perf_par_tab)
```

```
##   threshold       tpr
## 1 1.0000000 0.1071429
## 2 0.9715527 0.1785714
## 3 0.9582799 0.3214286
## 4 0.9525862 0.3392857
## 5 0.9041960 0.3571429
##
## ---
##         threshold tpr
## 96  1.355278e-05   1
## 97  1.306356e-05   1
## 98  1.288816e-05   1
## 99  4.371464e-06   1
## 100 4.275271e-06   1
## 101 0.000000e+00   1
```

```
# Conta o tempo de execução - RF
end.time <- Sys.time()
time.taken_rf_paral <- end.time - start.time
time.taken_rf_paral
```

```
## Time difference of 4.681985 mins
```

```
###################
# 3. GBM
# Inicia o contador do tempo de execução
start.time <- Sys.time()

## COLUMN
gbm_fit_par_col <- h2o.gbm(y = "classe",
                    training_frame = treino_col,
                    model_id = "gbm_fit_par_col",
                    validation_frame = validacao_col,
                    ntrees = 50,    # lembrar de colocar 500 arvores
                    balance_classes = TRUE,
                    seed = 1)
```

```
##
  |
  |                                                                  |   0%
  |
```

```
|=                                                                |   2%
|
|===                                                              |   4%
|
|====                                                             |   6%
|
|=====                                                            |   8%
|
|======                                                           |  10%
|
|========                                                         |  12%
|
|=========                                                        |  14%
|
|==========                                                       |  16%
|
|============                                                     |  18%
|
|=============                                                    |  20%
|
|==============                                                   |  22%
|
|================                                                 |  24%
|
|=================                                                |  26%
|
|==================                                               |  28%
|
|====================                                             |  30%
|
|=====================                                            |  32%
|
|======================                                           |  34%
|
|=======================                                          |  36%
|
|=========================                                        |  38%
|
|==========================                                       |  40%
|
|===========================                                      |  42%
|
|=============================                                    |  44%
|
|==============================                                   |  46%
|
|===============================                                  |  48%
|
|================================                                 |  50%
|
|==================================                               |  52%
|
|===================================                              |  54%
|
```

```
  |====================================                             |  56%
  |
  |======================================                           |  58%
  |
  |=======================================                          |  60%
  |
  |========================================                         |  62%
  |
  |==========================================                       |  64%
  |
  |===========================================                      |  66%
  |
  |============================================                     |  68%
  |
  |==============================================                   |  70%
  |
  |===============================================                  |  72%
  |
  |================================================                 |  74%
  |
  |=================================================                |  76%
  |
  |===================================================              |  78%
  |
  |====================================================             |  80%
  |
  |=====================================================            |  82%
  |
  |=======================================================          |  84%
  |
  |========================================================         |  86%
  |
  |=========================================================        |  88%
  |
  |==========================================================       |  90%
  |
  |============================================================     |  92%
  |
  |=============================================================    |  94%
  |
  |==============================================================   |  96%
  |
  |================================================================ |  98%
  |
  |=================================================================| 100%
```

```r
# save the model
#h2o.saveModel(object=gbm_fit_par_col, path=getwd(), force=TRUE)

# Let's compare the performance of the two GBMs.
gbm_perf_par_col <- h2o.performance(model = gbm_fit_par_col, newdata = teste_col)

# Print model performance
gbm_perf_par_col
```

```
## H2OBinomialMetrics: gbm
##
## MSE:  0.0001259307
## RMSE:  0.01122189
## LogLoss:  0.0004461843
## Mean Per-Class Error:  0
## AUC:  1
## Gini:  1
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##              0   1    Error        Rate
## 0      1852774   0 0.000000  =0/1852774
## 1            0 252 0.000000      =0/252
## Totals 1852774 252 0.000000  =0/1853026
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.037709 1.000000   0
## 2                       max f2  0.037709 1.000000   0
## 3                 max f0point5  0.037709 1.000000   0
## 4                 max accuracy  0.037709 1.000000   0
## 5                max precision  0.037709 1.000000   0
## 6                   max recall  0.037709 1.000000   0
## 7              max specificity  0.037709 1.000000   0
## 8             max absolute_mcc  0.037709 1.000000   0
## 9   max min_per_class_accuracy  0.037709 1.000000   0
## 10 max mean_per_class_accuracy  0.037709 1.000000   0
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

```r
h2o.auc(gbm_perf_par_col)
```

```
## [1] 1
```

```r
h2o.F1(gbm_perf_par_col)
```

```
##      threshold           f1
## 1 3.770917e-02 1.0000000000
## 2 4.168822e-07 0.0002719506
```

```r
h2o.precision(gbm_perf_par_col)
```

```
##      threshold    precision
## 1 3.770917e-02 1.0000000000
## 2 4.168822e-07 0.0001359938
```

```r
h2o.recall(gbm_perf_par_col)
```

```
##      threshold tpr
## 1 3.770917e-02   1
## 2 4.168822e-07   1
```

```r
## TABLE
gbm_fit_par_tab <- h2o.gbm(y = "classe",
                    training_frame = treino_tab,
                    model_id = "gbm_fit_par_tab",
                    validation_frame = validacao_tab,
```

```r
                    ntrees = 50,    # lembrar de colocar 500 arvores
                    balance_classes = TRUE,
                    seed = 1)
```

```
##
  |
  |                                                                      |   0%
  |
  |=====                                                                 |   8%
  |
  |==========                                                            |  16%
  |
  |=============                                                         |  20%
  |
  |=================                                                     |  26%
  |
  |================================                                      |  48%
  |
  |==============================================================        |  88%
  |
  |======================================================================| 100%
```

```r
# save the model
#h2o.saveModel(object=gbm_fit_par_tab , path=getwd(), force=TRUE)

# Let's compare the performance of the two GBMs.
gbm_perf_par_tab <- h2o.performance(model = gbm_fit_par_tab, newdata = teste_tab)

# Print model performance
gbm_perf_par_tab
```

```
## H2OBinomialMetrics: gbm
##
## MSE:  0.0005559258
## RMSE:  0.02357808
## LogLoss:  0.001432985
## Mean Per-Class Error:  0
## AUC:  1
## Gini:  1
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##            0  1    Error      Rate
## 0      69453  0 0.000000  =0/69453
## 1          0 56 0.000000     =0/56
## Totals 69453 56 0.000000  =0/69509
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                        metric threshold    value idx
## 1                      max f1  0.169318 1.000000   0
## 2                      max f2  0.169318 1.000000   0
## 3                max f0point5  0.169318 1.000000   0
## 4                max accuracy  0.169318 1.000000   0
## 5               max precision  0.169318 1.000000   0
## 6                  max recall  0.169318 1.000000   0
## 7             max specificity  0.169318 1.000000   0
```

```
## 8             max absolute_mcc  0.169318 1.000000   0
## 9   max min_per_class_accuracy  0.169318 1.000000   0
## 10 max mean_per_class_accuracy  0.169318 1.000000   0
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F
```

```r
h2o.auc(gbm_perf_par_tab)
```

```
## [1] 1
```

```r
h2o.F1(gbm_perf_par_tab)
```

```
##      threshold          f1
## 1 1.693178e-01 1.000000000
## 2 2.168401e-06 0.001610005
```

```r
h2o.precision(gbm_perf_par_tab)
```

```
##      threshold    precision
## 1 1.693178e-01 1.0000000000
## 2 2.168401e-06 0.0008056511
```

```r
h2o.recall(gbm_perf_par_tab)
```

```
##      threshold tpr
## 1 1.693178e-01   1
## 2 2.168401e-06   1
```

```r
# Conta o tempo de execução - GBM
end.time <- Sys.time()
time.taken_gbm_paral <- end.time - start.time
time.taken_gbm_paral
```

```
## Time difference of 3.006662 mins
```

```r
###########
# Desliga o H2O
h2o.shutdown(prompt = FALSE)
```

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

```r
# Remove todos os objetos e faz garbage collector
#rm(list = ls())
gc()
```

```
##            used  (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells  2080968 111.2   13610894   727.0   43523207  2324.4
## Vcells 21091229 161.0 1382472951 10547.5 5718066062 43625.4
```