Descrição do Método

Danusio Guimarães

03/04/2021

Contents

1 INTRODUÇÃO **1**

2 BIBLIOTECAS UTILIZADAS **1**

3 FUNÇÕES AUXILIARES **2**

4 ETL DOS DADOS **5**
4.1 Transformação de .csv para .fst 5
4.2 Carregamento dos Arquivos .fst 5
4.3 Cadastro 5
4.4 Segmentação por Cluster 7
4.5 Agrupamento 11
4.6 Segmentação para treino 15
4.7 EDA Básica 15
4.8 TPV Mensais 17

5 MONTAGEM DOS DATASETS DE TREINO **21**
5.1 Formatos para Treinamentos e Predições 22

6 TREINAMENTOS E PREDIÇÕES **22**

7 VARIÁVEIS POSSIVELMENTE ÚTEIS NA PREVISÃO **27**

1 INTRODUÇÃO

Este documento procura explicar a metodologia utilizada para a previsão de TPV proposta pelo Stone Data Challenge. A implementação foi feita na linguagem R, usando a IDE RStudio.

A maior parte da Análise Exploratória dos Dados (EDA) será feita *inline* em cada passo de execução, e não em uma seção a parte, pois a EDA é normalmente utilizada para gerar *insights* e entender os dados, em um processo recorrente e iterativo. Todas as análises desse tipo estarão precedidas de **EDA**, para melhor legibilidade.

Alguns procedimentos têm compilação demorada. Por isso, constam alguns comandos `load` carregando os resultados desses processos mais dispendiosos, com seu código gerador logo abaixo.

2 BIBLIOTECAS UTILIZADAS

As bibliotecas utilizadas para executar os procedimentos são:

- `fst`: permite manipular arquivos em formato .fst, que têm escrita e leitura mais rápidas que .csv;

- `magrittr`: permite o uso do operador *pipe* (`%>%`);
- `caret`: *framework* para contruir e testar modelos de *machine learning*;
- `stringr`: permite manipular texto de uma forma rápida e intuitiva;
- `modeest`: provê estimadores de moda estatística;
- `parallel` e `doParallel`: permite computação *multithread* (o R, por padrão, só utiliza 1 núcleo de processamento);
- `FSelector`: pacote para seleção de preditores;
- `fastDummies`: permite a criação de variáveis binárias geradas a partir de preditores categóricos.
- `factoextra`: usado para obter a "tendência de agrupamento" (o quão "agrupável" é o *dataset*) das observações.
- `cluster`: usado para calcular a silhueta do agrupamento.
- `clValid`: usado para calcular o Índice de Dunn do agrupamento.

```
library(fst)
library(magrittr)
library(caret)
library(caretEnsemble)
library(stringr)
library(modeest)
library(parallel)
library(doParallel)
library(FSelector)
library(fastDummies)
library(factoextra)
library(cluster)
library(clValid)
```

3 FUNÇÕES AUXILIARES

Por sua recorrência e/ou complexidade, alguns procedimentos foram transformados em funções:

- `attrNA`: faz imputação de dados faltantes (`NA`) via regressão exponencial. Será aplicado aos dados temporais, utilizando como variável independentes da regressão linear um indexador ordinal (no caso em tela, será relativo ao mês do faturamento). Por exemplo, se os faturamentos disponíveis são:

$$TPV = \{JUL = 100, JUN = 120, MAI = NA, ABR = 110, MAR = 150, FEV = 80, JAN = NA\}$$

Os valores de Maio e Janeiro serão estimados por uma regressão exponencial de `TPV ~ x={1,2,3,4,5,6,7}`, já que há 7 meses no total. No caso real, essa regressão será feita em um vetor de 1 a 37. A regressão exponencial foi escolhida por sua velocidade, fundamental dada a grande quantidade instâncias presentes, e pelo formato mais próximo à Normal dos TPV quando feita uma tranformação logarítmica (**EDA** - veja na seção 4.8.1).

A tranformação logarítmica proposta é a seguinte, que permite lidar com valores negativos:

$$y_{log} = \ln\left(y - y_{min} + 1\right)$$

O valor y_{log} será a variável dependente da regressão linear com o vetor $x = \{1, 2, 3..., 37\}$.

- `featSel`: faz a seleção de preditores combinando 3 metodologias: *oneR*, *information gain* e *Chi-squared*. Os valores de importância dados por cada metodologia comporão uma média e serão escolhidos aqueles atributos com média igual ou maior ao quantil 75% (25% maiores médias de importância). Além disso, o número de *features* foi limitado a 15, para evitar excesso de ajuste do modelo treinado.

- `em` e `emProj`: permitem, repectivamente, calcular os valores ajustados de uma reressão exponencial e prjetar `n` valores à frente por meio de uma regressão exponencial.
- `euclid`: calcula a distância euclidiana entre dois vetores.
- `knnImp`: dado um número de vizinhos `k`, encontra as `k` observações mais próximas, dentre as observações sem *missing values*, de cada observação com valores faltantes, calcula a média ponderada pelo inverso da distância dessas `k` instâncias e substitui os valores `NA` pelos correspondentes nesse vetor de médias ponderadas. Retorna um `dataset` sem valores faltantes.

```
transfLog <- function(x) log(x - min(x,na.rm = T)+1) %>% as.numeric
```

```
em <- function(y){
  x <- 1:length(y)
  ymin <- min(y,na.rm = T)
  ymax <- max(y,na.rm = T)

  y1 <- transfLog(y)

  A <- cov(y1,x,use = "c")/var(x,na.rm = T)
  b <- mean(y1,na.rm = T) - A*mean(x)

  yhat <- exp(A*x+b) + ymin - 1

  yhat
}
```

```
emProj <- function(y,n){
  if (sum(!is.na(y))==0) {
    return(rep(NA,n))
  }

  X <- 0:(1-n)

  x <- 1:length(y)
  ymin <- min(y,na.rm = T)
  ymax <- max(y,na.rm = T)

  y1 <- transfLog(y)

  A <- cov(y1,x,use = "c")/var(x,na.rm = T)
  b <- mean(y1,na.rm = T) - A*mean(x)

  Y <- exp(A*X+b) + ymin - 1

  return(Y)
}
```

```
attrNA <- function(y){
  if (sum(!is.na(y))==0) {
    return(y)
  }

  out <- ifelse(is.na(y),em(y),y) %>% as.numeric
}
```

```
euclid <- function(x,Y){
  dist <- NULL
  for (i in 1:nrow(Y)) {
    y <- Y[i,]
    d <- ((x - y)^2) %>% sum(na.rm = T) %>% sqrt
    dist <- c(dist,d)
  }

  dist
}
```

```
knnImp <- function(X,k){
  X <- as.matrix(X)
  N <- nrow(X)
  comp_cases <- complete.cases(X)

  if (sum(comp_cases) == N) {
    return(X)
  }

  comp_inst <- (1:N)[complete.cases(X)]

  X1 <- X[comp_inst,]

  out <- NULL
  for (i in 1:N) {
    if (i %in% comp_inst) {
      out <- rbind(out,X[i])
    }else{
      d <- euclid(X[i,],X1)
      pos <- order(d)[1:k]
      w <- matrix(1/d[pos],nrow = 1)/sum(1/d[pos])

      wavg <- (w %*% X1[pos,]) %>% as.numeric

      X_replace <- ifelse(is.na(X[i,]),
                          wavg,X[i,])
      out <- rbind(out,X_replace)
    }
  }

  out
}
```

```
featSel <- function(df){
  imp1 <- oneR(outcome ~ .,df)
  imp2 <- information.gain(outcome ~ .,df)
  imp3 <- chi.squared(outcome ~ .,df)

  feat_imp <- ((imp1+imp2+imp3)/3) %>%
    apply(2,sort,decreasing=T)

  mi_imp <- quantile(feat_imp,0.75,names = F)
  pred_sel <- rownames(feat_imp)[feat_imp[,1]>=mi_imp]
```

```
  # limitação a 15 preditores
  pred_sel <- pred_sel[1:min(15,length(pred_sel))]

  pred_sel
}
```

4 ETL DOS DADOS

4.1 Transformação de .csv para .fst

```
# computação em paralelo
Mycluster = makeCluster(detectCores()-2,
                        setup_strategy = "sequential")
registerDoParallel(Mycluster)

#-----------------------------------------------------\
tpv_mes_csv <- read.csv("tpv-mensais-treinamento.csv")
cadastro_csv <- read.csv("cadastrais.csv")

write_fst(tpv_mes_csv,path = "tpv-mensais-treinamento.csv")
write_fst(cadastro_csv,"cadastrais.fst")

#-----------------------------------------------------\

# setup inicial de processamento
stopCluster(Mycluster)
registerDoSEQ()
```

4.2 Carregamento dos Arquivos .fst

```
tpv_mes <- read_fst("tpv-mensais-treinamento.fst")
cadastro <- read_fst("cadastrais.fst")
```

```
# id's de clientes
clientes <- unique(tpv_mes$id)
```

4.3 Cadastro

```
# variável de cópia, por segurança
cad1 <- cadastro
```

- Tranformação da variável com o porte da empresa para "fator ordenado", já que existe uma relação entre as magnitudes de cada *level*. Além disso, o identificador de segmento MCC deve ser interpretado como um fator (variável categórica), não como um número:

```
# porte com fator ordenado
cad1$porte <- ordered(cadastro$porte,
                      levels = c("0-2.5k","2.5k-5k","5k-10k",
                                 "10k-25k","25k-50k",
                                 "50k-100k","100k-500k","500k+"))
# MCC como fator
cad1$MCC <- as.factor(cad1$MCC)
```

- Eliminação de valores faltantes, provavelmente resultantes de erros de registro. Não podem ser eliminados, entretanto, se excluírem algum cliente da base de dados:

```
# eliminação de NA's
cad2 <- cad1 %>% na.omit
cad2$tipo_documento <- cad2$tipo_documento %>%
  as.character %>% as.factor
```

Para checar se todos os clientes ainda constam na base `cad2`:

```
# nº de clientes que não estão presentes na features 'id' de 'cad2'
sum(!(clientes %in% cad2$id))
```

```
>  [1] 0
```

- Tranformação do atributo referente à data de inclusão do cliente na base de dados para formato de data. Além disso, eliminação da `feature` relativa à data da primeira transação, já que essa informação pode ser tirada dos dados de TPV da seção anterior:

```
# mudança de formato de StoneCreatedDate
cad2$StoneCreatedDate <- cad2$StoneCreatedDate %>% as.Date
# eliminação de StoneFirstTransactionDate
cad2$StoneFirstTransactionDate <- NULL
```

- Criação da variável `Ticket`, cuja informação está aglutinada no atributo `persona`. Separar essas duas informações pode facilitar a construção dos modelos de predição:

```
# criação da variável Ticket
tickets <- (cad2$persona %>% as.character %>%
  str_split(" ",simplify = T))[,6:7] %>%
  apply(1,paste0,collapse=" ")

tickets[tickets == " "] <- "Outro"
cad2$Ticket <- tickets %>% as.factor
```

- Identificação e eliminação de múltiplas instâncias para a mesma ID de cliente, que deve ser única. Será contado o mais recente registro no banco de dados, exceto pelas variáveis `StoneCreateDate`, que usará o registro mais antigo, e `Estado`, que usará a moda dos registros. Neste último, caso haja empate de ocorrência, será escolhido o registro mais recente:

```
# computação em paralelo
Mycluster = makeCluster(detectCores()-2,
                        setup_strategy = "sequential")
registerDoParallel(Mycluster)

# contagem de registros duplos
df_dupl <- cont <- NULL
for (i in 1:length(clientes)) {
  s <- sum(cad2$id == clientes[i],na.rm = T)

  if (s>1) {
    cont <- c(cont,clientes[i])
    # eliminação de registros duplos
    df <- subset(cad2,id==clientes[i]) %>% unique
    x <- tail(df,1)
    x$StoneCreatedDate <- df$StoneCreatedDate[1]
    x$Estado <- mfv(df$Estado) %>% tail(1)

    df_dupl <- rbind(df_dupl,
```

```
                    x)
  }
}

# setup inicial de processamento
stopCluster(Mycluster)
registerDoSEQ()
```

- Consolidação dos dados após a eliminação de instâncias múltiplas, com o posterior check de presença de todos os clientes na *data frame* final:

```
# instâncias relativas aos clientes em 'tpv_mes'
cad3 <- cad2
cad3 <- cad3[!(cad3$id %in% cont),]
cad3 <- rbind(cad3,
              df_dupl)
cad3 <- cad3[cad3$id %in% clientes,]
cad3 <- cad3[order(cad3$id),]

cad3$MacroClassificacao <- as.character(cad3$MacroClassificacao)
cad3$MacroClassificacao[cad3$MacroClassificacao == ""] <- "Outro"
cad3$MacroClassificacao <- as.factor(cad3$MacroClassificacao)
cad3$MCC <- as.factor(cad3$MCC)

rm(cadastro,cad1,cad2)
```

```
sum(!(clientes %in% cad3$id))
```

```
>  [1] 0
```

4.4 Segmentação por Cluster

Uso dos dados de cadastro para construir *clusters*, sobre os quais serão treinados os modelos. A hipótese aqui assumida é a de que reunir instâncias semelhantes otimiza o desempenho dos modelos treinados; a tarefa não-supervisionada foi escolhida por permitir o aparecimento de *clusters* mais "naturais" que aqueles contruídos manualmente, por macroclassificação ou tipo de documento, por exemplo.

O algoritmo escolhido foi o kNN com distância euclidiana.

4.4.1 Variáveis dummy para cadastro

- (**EDA**) Nº de *levels* para cada *feature* do *dataset* `cad3`:

```
cbind(names(cad3),
      cad3 %>% sapply(class) %>% as.character)
```

```
>        [,1]                 [,2]
>   [1,] "id"                 "integer"
>   [2,] "StoneCreatedDate"   "Date"
>   [3,] "MCC"                "factor"
>   [4,] "MacroClassificacao" "factor"
>   [5,] "segmento"           "character"
>   [6,] "sub_segmento"       "character"
>   [7,] "persona"            "character"
>   [8,] "porte"              "c(\"ordered\", \"factor\")"
>   [9,] "TPVEstimate"        "numeric"
>  [10,] "tipo_documento"     "factor"
```

```
>  [11,] "Estado"             "character"
>  [12,] "Ticket"             "factor"
```

À exceção de `id`, `StoneCreatedDate` e `TPVEstimate`, todos os preditores são fatores. A variável `porte` é um fator ordenado, e por isso é necessária sua transformação para apenas fator, a fim de extrair variáveis binárias:

```
cad4 <- cad3
cad4$porte <- cad3$porte %>% as.character %>% as.factor
class(cad4$porte)
```

```
>  [1] "factor"
```

É interessante verificar o número de `levels` de cada variável para definir a viabilidade de criar variáveis binárias a parte dela (muitas categorias torna desaconselhável esse processo):

```
cad4[,-c(1,2,9)] %>% lapply(levels) %>% sapply(length,simplify = T)
```

```
>                 MCC MacroClassificacao           segmento       sub_segmento
>                 261                  9                  0                  0
>             persona              porte     tipo_documento             Estado
>                   0                  8                  3                  0
>              Ticket
>                   4
```

`MCC` possui número excessivo de categorias, bem como `sub_segmento`. Entretanto, `Estado` deve apresentar inconsistência de registro, já que deve ser limitado a 27 categorias, no Brasil:

```
levels(cad4$Estado)
```

```
>  NULL
```

Vamos verificar quais estados não estão representados por siglas:

```
estados <- levels(cad4$Estado) %>% unique
estados[str_count(estados)>2]
```

```
>  NULL
```

Goiás, Paraná e São Paulo estão com duas grafias:

```
cad4$Estado <- str_replace_all(cad4$Estado %>% as.character,
                               "Goias","Goiás") %>%
  as.factor
cad4$Estado <- str_replace_all(cad4$Estado %>% as.character,
                               "Parana","Paraná") %>%
  as.factor
cad4$Estado <- str_replace_all(cad4$Estado %>% as.character,
                               "Sao Paulo","São Paulo") %>%
  as.factor

estados <- levels(cad4$Estado) %>% unique
estados[str_count(estados)>2]
```

```
>   [1] "Acre"               "Alagoas"            "Amapá"
>   [4] "Amazonas"           "Bahia"              "Ceará"
>   [7] "Distrito Federal"   "Espírito Santo"     "Goiás"
>  [10] "Maranhão"           "Mato Grosso"        "Mato Grosso do Sul"
>  [13] "Minas Gerais"       "Pará"               "Paraíba"
>  [16] "Paraná"             "Pernambuco"         "Piauí"
>  [19] "Rio de Janeiro"     "Rio Grande do Norte" "Rio Grande do Sul"
```

```
>  [22] "Rondônia"            "Roraima"             "Santa Catarina"
>  [25] "São Paulo"           "Sergipe"             "Tocantins"
```

Agora, basta substituir essas ocorrências por suas siglas:

```
siglas <- c("AC","AL","AP","AM","BA","CE","DF","ES","GO","MA","MT","MS","MG",
            "PA","PB","PR","PE","PI","RJ","RN","RS","RO","RR","SC","SP","SE","TO")
estados_ext <- estados[str_count(estados)>2] %>% as.character

for (i in 1:length(siglas)) {
  cad4$Estado <- str_replace_all(cad4$Estado %>% as.character,
                                 estados_ext[i],siglas[i]) %>%
  as.factor
}

levels(cad4$Estado)
```

```
>   [1] ""          "AC"        "AL"        "AM"        "AP"        "BA"
>   [7] "CE"        "DF"        "ES"        "GO"        "MA"        "MG"
>  [13] "MS"        "MT"        "MT do Sul" "PA"        "PB"        "PE"
>  [19] "PI"        "PR"        "Rj"        "RJ"        "RN"        "RO"
>  [25] "RR"        "RS"        "SC"        "SE"        "Sp"        "SP"
>  [31] "TO"
```

Resolvendo o problema de "MT do Sul", das letras minúsculas e dos valores faltantes:

```
cad4$Estado <- str_replace_all(cad4$Estado %>% as.character,
                               "MT do Sul","MS") %>%
  casefold(upper = T) %>% as.factor

cad4$Estado <- as.character(cad4$Estado)
cad4$Estado[cad4$Estado==""] <- "unknown"
cad4$Estado <- as.factor(cad4$Estado)

summary(cad4$Estado) %>% sort(decreasing = T)
```

```
>       SP       RJ       MG       PR       SC       RS       BA       GO       PE       DF
>    54201    20882    19047    18527    15177    13103     9830     7863     6234     4868
>       CE       ES       PA       MT       RN       MA       PB       SE unknown       AL
>     4687     4521     3248     2844     2723     2425     2222     2201     2149     1638
>       MS       AM       RO       PI       TO       AP       RR       AC
>     1612     1405     1034      944      788      597      549      514
```

- Variável `segmento`:

```
levels(cad4$segmento)
```

```
>  NULL
```

Em uma primeira abordagem, consideraremos que essa informação está abrangida por `Macroclassificação`. Os preditores não utilizados, até agora, para a criação das variáveis binárias são: 1,2,3,5,6 e 9.

- Variável `persona`:

```
levels(cad4$persona)
```

```
>  NULL
```

Consideraremos também que essas informações podem ser acessadas por outras *features*, adicionando a variável 7 à lista de não-utilizadas no processo de "dummyzation".

- Variáveis *dummy*:

```
cad_dummy <- dummy_cols(cad4[,-c(1:3,5:7,9)],
                        remove_first_dummy = T,
                        remove_selected_columns = T)
names(cad_dummy)
```

```
>   [1] "MacroClassificacao_Bens duráveis"
>   [2] "MacroClassificacao_Outro"
>   [3] "MacroClassificacao_Posto"
>   [4] "MacroClassificacao_Serviços"
>   [5] "MacroClassificacao_Serviços recorrentes"
>   [6] "MacroClassificacao_Supermercado/Farmácia"
>   [7] "MacroClassificacao_Varejo"
>   [8] "MacroClassificacao_Viagens e entretenimento"
>   [9] "porte_100k-500k"
>  [10] "porte_10k-25k"
>  [11] "porte_2.5k-5k"
>  [12] "porte_25k-50k"
>  [13] "porte_500k+"
>  [14] "porte_50k-100k"
>  [15] "porte_5k-10k"
>  [16] "tipo_documento_PF"
>  [17] "tipo_documento_PJ"
>  [18] "Estado_AL"
>  [19] "Estado_AM"
>  [20] "Estado_AP"
>  [21] "Estado_BA"
>  [22] "Estado_CE"
>  [23] "Estado_DF"
>  [24] "Estado_ES"
>  [25] "Estado_GO"
>  [26] "Estado_MA"
>  [27] "Estado_MG"
>  [28] "Estado_MS"
>  [29] "Estado_MT"
>  [30] "Estado_PA"
>  [31] "Estado_PB"
>  [32] "Estado_PE"
>  [33] "Estado_PI"
>  [34] "Estado_PR"
>  [35] "Estado_RJ"
>  [36] "Estado_RN"
>  [37] "Estado_RO"
>  [38] "Estado_RR"
>  [39] "Estado_RS"
>  [40] "Estado_SC"
>  [41] "Estado_SE"
>  [42] "Estado_SP"
>  [43] "Estado_TO"
>  [44] "Estado_unknown"
>  [45] "Ticket_Ticket Alto"
>  [46] "Ticket_Ticket Baixo"
>  [47] "Ticket_Ticket Medio"
```

No total, temos 47 variáveis binárias para o processo de agrupamento.

4.5 Agrupamento

- Tendência de Agrupamento:

(**EDA**) Avaliação da *tendência de agrupamento* dos dados por meio da estatística de Hopkins, que vai de 0 a 1. Quanto maior seu valor, mais *clusterizável* é o conjunto de dados, e o ideal é que fique acima de 0,5:

```
nsamples <- 500
ntry <- 20
inst_tot <- 1:nrow(cad_dummy)

# computação em paralelo
Mycluster = makeCluster(detectCores()-2,
                        setup_strategy = "sequential")
registerDoParallel(Mycluster)

t0 <- Sys.time()
hpk <- NULL
for (i in 1:ntry) {
  inst <- sample(inst_tot,size = nsamples+1,replace = F)
  tend_cl <- get_clust_tendency(cad_dummy[inst,],n=nsamples)
  hpk <- c(hpk,tend_cl$hopkins_stat)
}
t1 <- Sys.time()
# setup inicial de processamento
stopCluster(Mycluster)
registerDoSEQ()
```

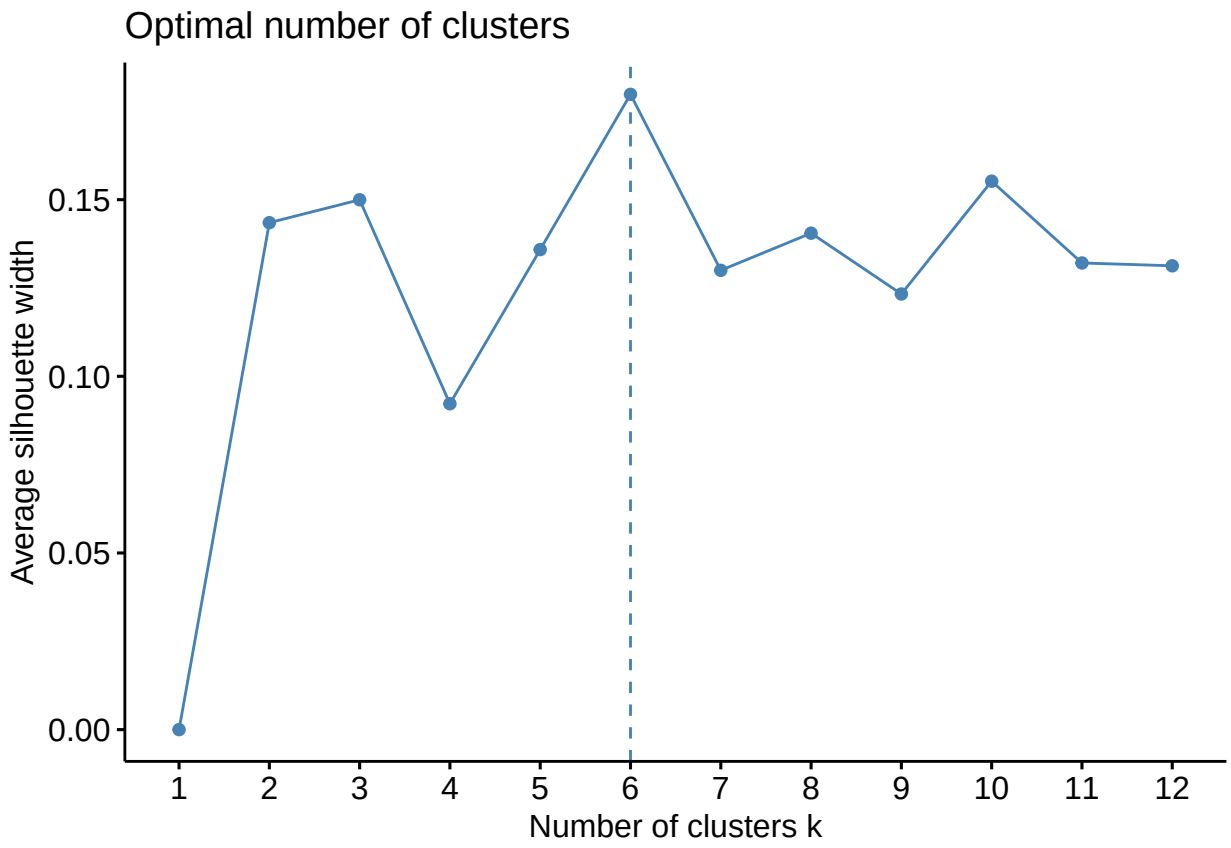
```
c(media = mean(hpk),
  desvio_padrao = sd(hpk))
```

```
>          media desvio_padrao
>    0.745584142   0.004026982
```

Com a estatística Hopkins acima de 0.5, vamos aceitar a hipótese de que o *dataset* `cad_dummy` tem boa tendência a clusterização.

- Nº ótimo de clusters: o critério será o maior valor da silhueta média do agrupamento. Abaixo, temos um exemplo para as primeiras 5000 instâncias, com o valor de silhueta plotado contra o número de grupos a serem gerados:

```
# exemplo
fviz_nbclust(cad_dummy[1:5000,],
             FUNcluster = kmeans,
             method = "silhouette",
             k.max = 12,verbose = F)
```



A computação para determinar o nº ótimo de *clusters* é:

```
nsamples <- 5000
ntry <- 41
inst_tot <- 1:nrow(cad_dummy)

# computação em paralelo
Mycluster = makeCluster(detectCores()-2,
                        setup_strategy = "sequential")
registerDoParallel(Mycluster)

t0 <- Sys.time()
kopt <- max_silh <- NULL
for (i in 1:ntry) {
  inst <- sample(inst_tot,size = nsamples,replace = F)
  optCl <- fviz_nbclust(cad_dummy[inst,],
                        FUNcluster = kmeans,
                        method = "silhouette",
                        k.max = 12,verbose = F)$data
  kopt <- c(kopt,
            which.max(optCl[,2]))
  max_silh <- c(max_silh,
                max(optCl[,2]))

}
t1 <- Sys.time()
# setup inicial de processamento
```

```
stopCluster(Mycluster)
registerDoSEQ()
```

```
kopt
```

```
>   [1] 5 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
>  [39] 7 7 7
```

```
mean(kopt)
```

```
>  [1] 6.95122
```

```
sd(kopt)
```

```
>  [1] 0.3123475
```

```
mean(max_silh)
```

```
>  [1] 0.1555159
```

```
sd(max_silh)
```

```
>  [1] 0.00472231
```

Vemos uma clara convergência para o uso de 7 grupos, que será também o número de modelos a serem gerados para a predição de cada grupo.

4.5.1 K-means

Modelagem para 7 grupos:

```
k <- 7
```

```
cl_model <- kmeans(cad_dummy,7)
```

- Silhueta

```
nsamples <- 10000
ntry <- 20
inst_tot <- 1:nrow(cad_dummy)

# computação em paralelo
Mycluster = makeCluster(detectCores()-2,
                        setup_strategy = "sequential")
registerDoParallel(Mycluster)

t0 <- Sys.time()
sil_avg <- NULL
for (i in 1:ntry) {
  inst <- sample(inst_tot,size = nsamples,replace = F)
  sil <- silhouette(cl_model$cluster[inst],dist(cad_dummy[inst,]))
  sil_avg <- c(sil_avg,
               mean(sil[, "sil_width"]))
}
t1 <- Sys.time()

# setup inicial de processamento
stopCluster(Mycluster)
registerDoSEQ()
```

```
mean(sil_avg)
```

```
>  [1] 0.1201311
```

```
sd(sil_avg)
```

```
>  [1] 0.0007952851
```

A silhueta média ficou em torno de 0,12. Vejamos outras medida de qualidade, o Índice de Dunn:

```
# índice de Dunn
nsamples <- 5000
ntry <- 41
inst_tot <- 1:nrow(cad_dummy)

# computação em paralelo
Mycluster = makeCluster(detectCores()-2,
                        setup_strategy = "sequential")
registerDoParallel(Mycluster)

t0 <- Sys.time()
IDunn <- NULL
for (i in 1:ntry) {
  inst <- sample(inst_tot,size = nsamples,replace = F)
  ID <- dunn(Data = cad_dummy[1:nsamples,],
             clusters = cl_model$cluster[1:nsamples])
  IDunn <- c(IDunn,ID)
}
t1 <- Sys.time()
# setup inicial de processamento
stopCluster(Mycluster)
registerDoSEQ()
```

```
mean(IDunn)
```

```
>  [1] 0.3333333
```

```
sd(IDunn)
```

```
>  [1] 0
```

- Relação entre a média intra e a média inter:

```
mean(cl_model$withinss)/cl_model$betweenss
```

```
>  [1] 0.2988828
```

- Id's por grupo:

```
grupos <- cl_model$cluster

for (i in 1:k) {
  print(sum(grupos==i))
}
```

```
>  [1] 26430
>  [1] 19229
>  [1] 33095
>  [1] 29317
```

```
>  [1] 23210
>  [1] 31587
>  [1] 42965
```

As medidas de qualidade não são as ideais, mas aceitaremos que essa divisão é mais próxima da otimalidade que uma segmentação arbitrária ou da não-segmentação (esta última ainda conta com o problema de um aumento quadrático de tempo de processamento).

4.6 Segmentação para treino

As ID's de cada instância são armazenadas para posterior montagem dos *datasets* de treino e predição:

```
id_cluster <- list()
for (i in 1:k) {
  id_cluster[[i]] <- cad4$id[which(grupos==i)]
}
```

- Tranformação `porte` em fator ordenado novamente:

```
# porte com fator ordenado
cad4$porte <- ordered(cad4$porte,
                      levels = c("0-2.5k","2.5k-5k","5k-10k",
                                 "10k-25k","25k-50k",
                                 "50k-100k","100k-500k","500k+"))
```

4.7 EDA Básica

- Tipos dos preditores

```
tpv_mes %>% sapply(class,simplify = T)
```

```
>              id mes_referencia     TPV_mensal
>       "integer"      "integer"      "numeric"
```

Os dados de TPV são, naturalmente, numéricos. O mês de referência está, ainda, em um formato numérico.

```
cad4 %>% sapply(class,simplify = T)
```

```
>  $id
>  [1] "integer"
>
>  $StoneCreatedDate
>  [1] "Date"
>
>  $MCC
>  [1] "factor"
>
>  $MacroClassificacao
>  [1] "factor"
>
>  $segmento
>  [1] "character"
>
>  $sub_segmento
>  [1] "character"
>
>  $persona
>  [1] "character"
```

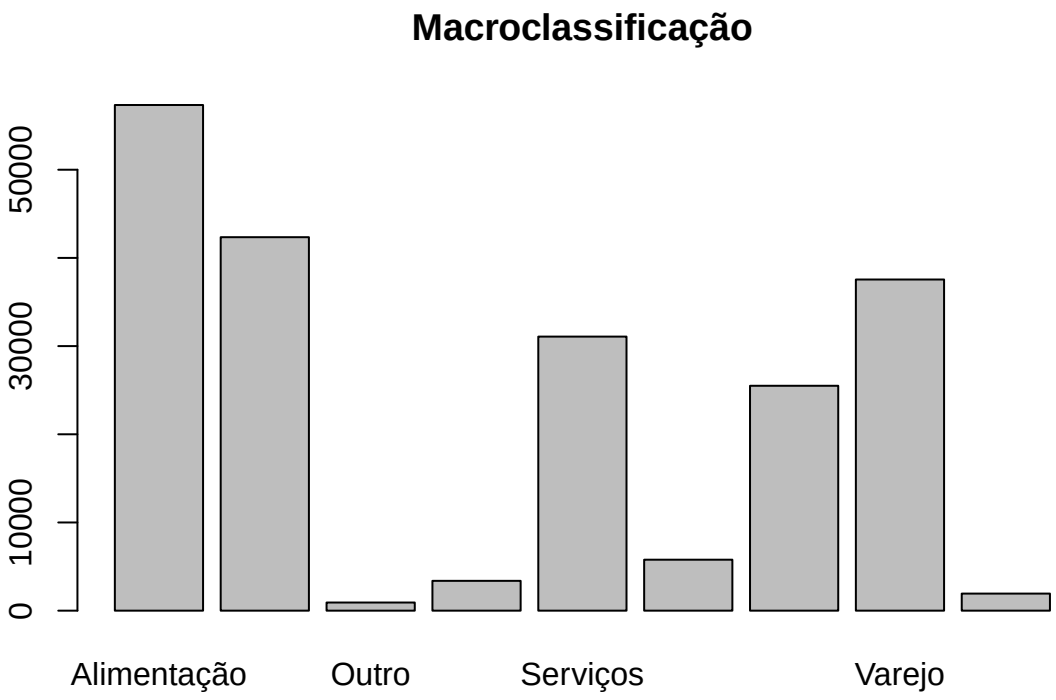
```
>
>  $porte
>  [1] "ordered" "factor"
>
>  $TPVEstimate
>  [1] "numeric"
>
>  $tipo_documento
>  [1] "factor"
>
>  $Estado
>  [1] "factor"
>
>  $Ticket
>  [1] "factor"
```

Apenas o TPV estimado está em um formato numérico de fato, o que sugere a necessidade de criação de variáveis binárias para esses preditores categórigos.

- Distribuição dos Dados

Vejamos a distribuição dos preditores aparentemente mais significativos:

```
plot(cad4$MacroClassificacao,
     main = "Macroclassificação")
```



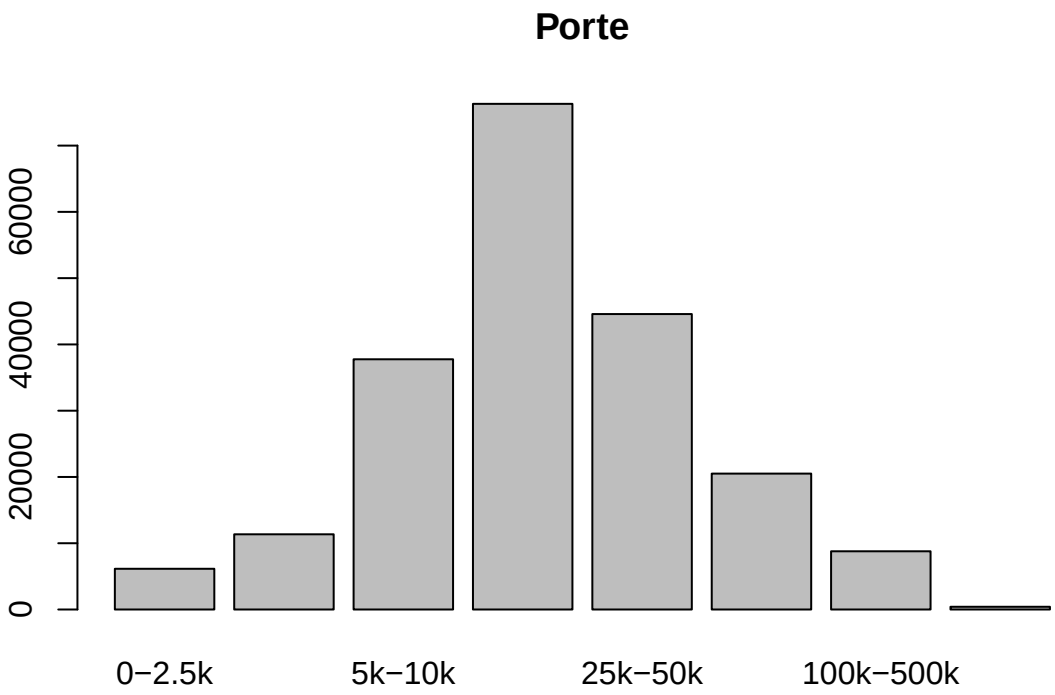

```
summary(cad4$MacroClassificacao) %>%
  sort(decreasing = T)
```

```
>              Alimentação            Bens duráveis                  Varejo
>                    57340                    42349                    37550
>                 Serviços    Supermercado/Farmácia     Serviços recorrentes
>                    31075                    25500                     5780
>                   Posto Viagens e entretenimento                    Outro
```

```
>                    3382                    1936                     921
```

Alimentação, Bens duráveis, Varejo e Serviços dominam os tipos de estabelecimentos.

```
plot(cad4$porte,
     main="Porte")
```




```
summary(cad4$porte) %>%
  sort(decreasing = T)
```

```
>    10k-25k    25k-50k     5k-10k   50k-100k    2.5k-5k  100k-500k     0-2.5k      500k+
>      76301      44580      37755      20507      11349       8784       6145        412
```

A maior parte das empresas tem um tamanho financeiro na faixa de 10 mil a 25 mil.

```
summary(cad4$Estado) %>%
  sort(decreasing = T) %>% head(10)
```

```
>     SP    RJ    MG    PR    SC    RS    BA    GO    PE    DF
>  54201 20882 19047 18527 15177 13103  9830  7863  6234  4868
```

São Paulo tem destacadamente a maior concentração de empresas do banco de dados.

4.8 TPV Mensais

- Extração das ID's únicas de cliente:

```
# variável de cópia, por segurança
tpv1 <- tpv_mes
# id's de clientes
clientes <- unique(tpv_mes$id)
```

- Tranformação do mês de referência para formato de data:

```
# mês de referência em formato de data
tpv1$mes_referencia <-  tpv_mes$mes_referencia %>%
  as.character %>% as.Date(format = "%Y%m%d")
```

- Criação de uma *dataset* no formato atributo-valor com os dados temporais de TPV. Cada mês de referência será equivalente a uma *feature*, com os clientes como instâncias:

```
# computação em paralelo
Mycluster = makeCluster(detectCores()-2,
                        setup_strategy = "sequential")
registerDoParallel(Mycluster)

# destemporalização de tpv1
datas <- unique(tpv1$mes_referencia) %>% sort(decreasing = T)
M <- matrix(NA,ncol = length(datas),nrow = length(clientes))
colnames(M) <- paste0("ref_",datas %>% as.character)
rownames(M) <- clientes

for (i in 1:ncol(M)) {
  df <- subset(tpv1,mes_referencia == datas[i])
  M[df$id %>% as.character,i] <- df$TPV_mensal
}

# setup inicial de processamento
stopCluster(Mycluster)
registerDoSEQ()
```

```
# consolidação
tpv2 <- data.frame(id = clientes,
                   M)
```

4.8.1 Atribuição de valores faltantes

- função `knnImputation`:

```
# computação em paralelo
Mycluster = makeCluster(detectCores()-2,
                        setup_strategy = "sequential")
registerDoParallel(Mycluster)

ini_time <-  Sys.time()

tpv3.1 <- ID <- NULL
for (i in 1:k) {
  ID <- c(ID,id_cluster[[i]])

  subdf <- tpv2[tpv2$id %in% id_cluster[[i]],-1]
  tpv3.1 <- rbind(tpv3.1,knnImp(subdf,k=5))
}

end_time <-  Sys.time()

# setup inicial de processamento
stopCluster(Mycluster)
registerDoSEQ()

tpv3.1 <- cbind(id = ID,
                tpv3.1)
rownames(tpv3.1) <- tpv3.1[,1] %>% as.character
```

```
tpv3.1 <- tpv3.1[order(tpv3.1[,1]),]
```

- função `attrNA`:

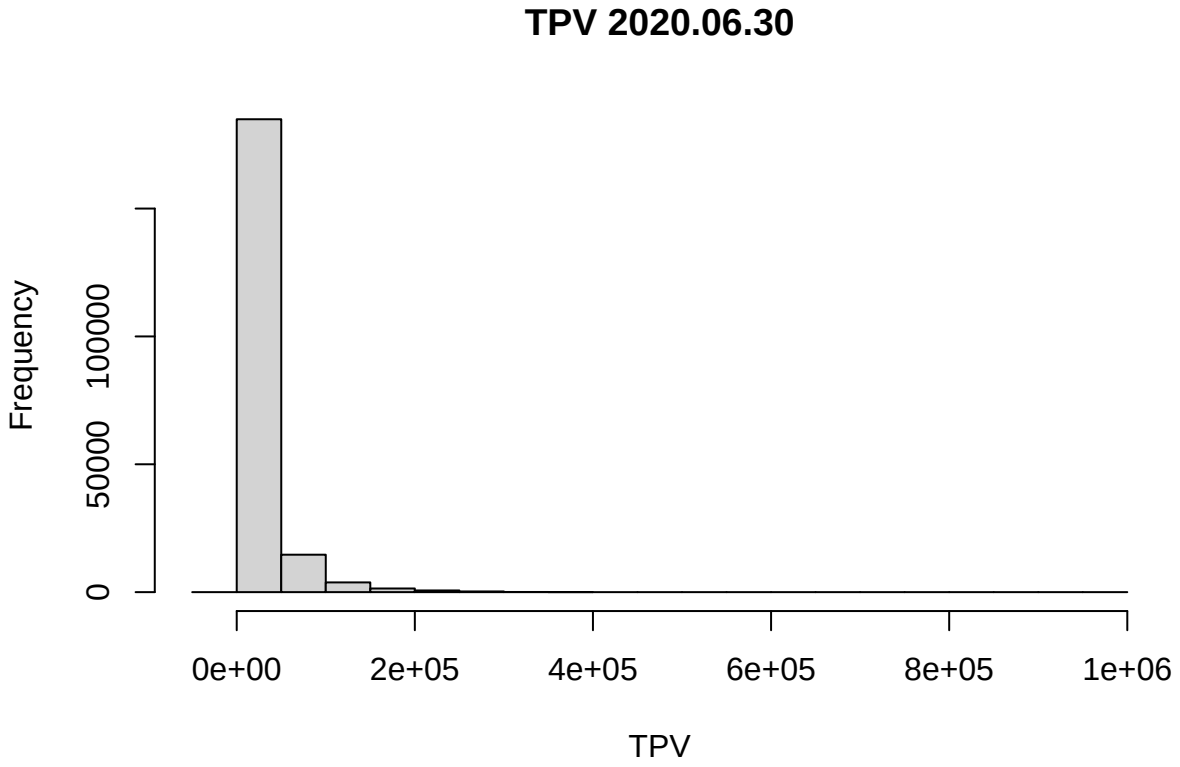
```
tpv3.2 <- tpv2
tpv3.2[,-1] <- tpv2[,-1] %>% apply(1,attrNA) %>% t
```

- Combinação das atribuições: para evitar um possível enviesamento dos modelos, e considerando que a imputação via kNN é mais acertada que a regressão exponencial, a combinação será uma média ponderada das duas imputações, com peso 70% para kNN e 30% para regressão exponencial:

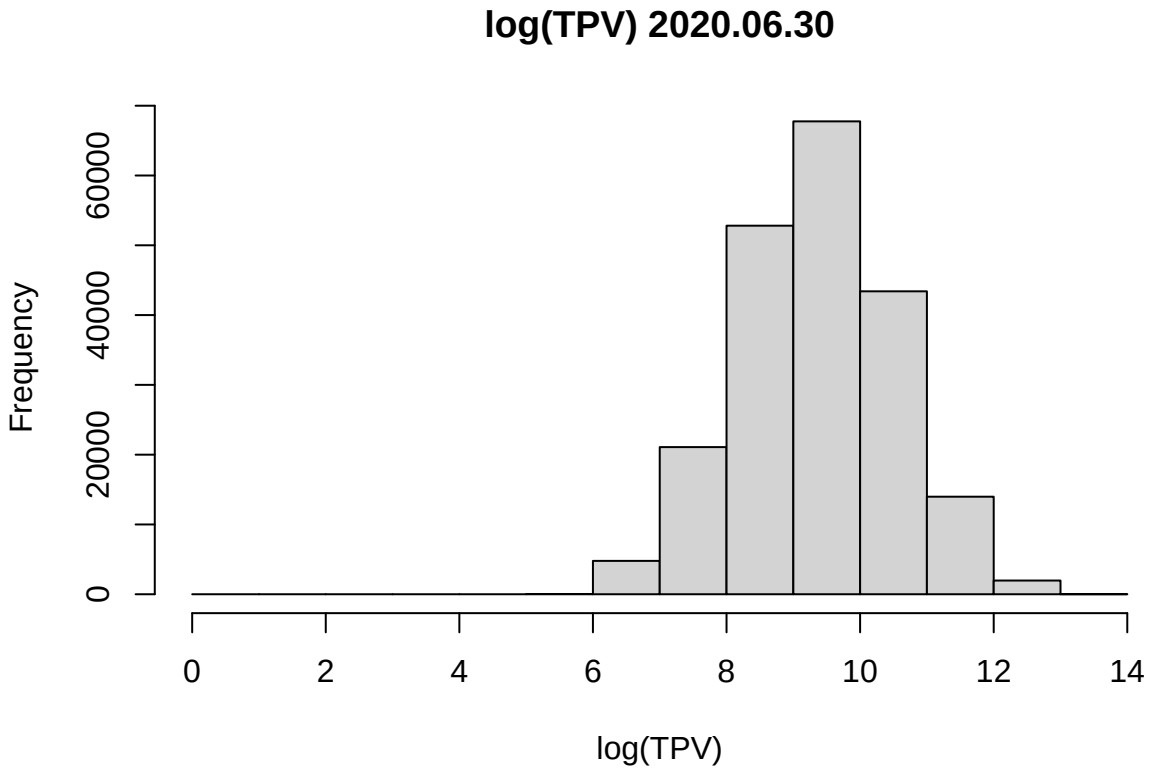
```
tpv3 <- 0.7*tpv3.1[,-1] + 0.3*tpv3.2[,-1]
pos_na <- which(tpv3 %>% is.na,arr.ind = T)

tpv3[pos_na] <- tpv3.1[pos_na]
tpv3 <- tpv3 %>% data.frame
tpv3 <- data.frame(id = clientes,
                   tpv3)
```

Observe-se a distribuição dos TPV para junho, citado na seção 3. O uso do logaritmo reduz a curtose dos dados, permitindo uma melhor detecção das nuances:




```
hist(log(tpv3$ref_2020.06.30-min(tpv3$ref_2020.06.30,na.rm = T) + 1),
     main = "log(TPV) 2020.06.30",xlab = "log(TPV)")
```



Esse padrão repete-se para os demais meses de referência.

4.8.2 Projeções de Preço

Foram criadas variáveis contendo as projeções de TPV para os meses de agosto a dezembro de 2020, a fim de registrar uma tendência dos faturamentos. O método usado foi de regressão exponencial.

- Projeções para 5 meses:

```
proj <- tpv3[,-1] %>% apply(MARGIN = 1,
                            FUN = emProj,
                            n=5) %>% t
colnames(proj) <- paste0(c("ago","set","out","nov","dez"),"20")
```

Para incluir tais projeções no modelo, serão computadas as previsões para julho considerando os dados até junho (equivalente à projeção de agosto, distância de 1 mês), maio (equivalente à projeção de setembro, distância de 2 meses) e assim por diante.

- Projeções para Julho 2020:

```
proj_tr <- cbind(
  (tpv3[,-c(1:2)] %>% apply(MARGIN = 1,
                            FUN = emProj,
                            n=2) %>% t)[,1],
  (tpv3[,-c(1:3)] %>% apply(MARGIN = 1,
                            FUN = emProj,
                            n=2) %>% t)[,2],
  (tpv3[,-c(1:4)] %>% apply(MARGIN = 1,
                            FUN = emProj,
                            n=3) %>% t)[,3],
  (tpv3[,-c(1:5)] %>% apply(MARGIN = 1,
                            FUN = emProj,
```

```
                              n=4) %>% t)[,4],
  (tpv3[,-c(1:6)] %>% apply(MARGIN = 1,
                              FUN = emProj,
                              n=5) %>% t)[,5]
)

colnames(proj_tr) <- colnames(proj)
```

5 MONTAGEM DOS DATASETS DE TREINO

- Dataset com variáveis numéricas

```
df_train_tpv <- data.frame(id = tpv3$id,
                           jul20=NA,
                           proj,
                           tpv3[,-1])
names(df_train_tpv)
```

```
>   [1] "id"             "jul20"          "ago20"          "set20"
>   [5] "out20"          "nov20"          "dez20"          "ref_2020.07.31"
>   [9] "ref_2020.06.30" "ref_2020.05.31" "ref_2020.04.30" "ref_2020.03.31"
>  [13] "ref_2020.02.29" "ref_2020.01.31" "ref_2019.12.31" "ref_2019.11.30"
>  [17] "ref_2019.10.31" "ref_2019.09.30" "ref_2019.08.31" "ref_2019.07.31"
>  [21] "ref_2019.06.30" "ref_2019.05.31" "ref_2019.04.30" "ref_2019.03.31"
>  [25] "ref_2019.02.28" "ref_2019.01.31" "ref_2018.12.31" "ref_2018.11.30"
>  [29] "ref_2018.10.31" "ref_2018.09.30" "ref_2018.08.31" "ref_2018.07.31"
>  [33] "ref_2018.06.30" "ref_2018.05.31" "ref_2018.04.30" "ref_2018.03.31"
>  [37] "ref_2018.02.28" "ref_2018.01.31" "ref_2017.12.31" "ref_2017.11.30"
>  [41] "ref_2017.10.31" "ref_2017.09.30" "ref_2017.08.31" "ref_2017.07.31"
```

- Variáveis de cadastro a serem incluídas no treinamento:

```
cad5 <- data.frame(outcome = df_train_tpv$ref_2020.07.31,
                   dummy_cols(cad4[,-c(1:3,5:7)],
                              remove_first_dummy = T,
                              remove_selected_columns = T))

mip <- featSel(cad5)
```

```
mip
```

```
>   [1] "porte"
>   [2] "TPVEstimate"
>   [3] "tipo_documento_PJ"
>   [4] "tipo_documento_PF"
>   [5] "Ticket_Ticket.Alto"
>   [6] "MacroClassificacao_Bens.duráveis"
>   [7] "Ticket_Ticket.Baixo"
>   [8] "Ticket_Ticket.Medio"
>   [9] "MacroClassificacao_Serviços"
>  [10] "MacroClassificacao_Supermercado.Farmácia"
>  [11] "MacroClassificacao_Posto"
```

As variáveis `porte`, `tipo_documento` e `Ticket` tiveram todas as suas categorias incluídas nas variáveis mais preditivas do TPV de julho de 2020. `TPVEstimate` também entrou no rol, mas `MacroClassificacao` teve

menos da metade de suas categorias incluídas. Assim, podemos montar o *dataset* com variáveis categóricas da seguinte forma:

```
df_train_cad <- cad4[,c("id", "TPVEstimate",
                        "porte","Ticket","tipo_documento")]
summary(df_train_cad)
```

```
>         id           TPVEstimate             porte                Ticket
>   Min.   :     1   Min.   :0.000e+00   10k-25k  :76301   Outro       :28598
>   1st Qu.: 51518   1st Qu.:9.000e+03   25k-50k  :44580   Ticket Alto :42277
>   Median :103005   Median :1.500e+04   5k-10k   :37755   Ticket Baixo:73701
>   Mean   :103048   Mean   :9.536e+04   50k-100k :20507   Ticket Medio:61257
>   3rd Qu.:154533   3rd Qu.:2.840e+04   2.5k-5k  :11349
>   Max.   :206330   Max.   :1.300e+10   100k-500k: 8784
>                                        (Other)  : 6557
>   tipo_documento
>   MEI: 39939
>   PF : 20586
>   PJ :145308
>
>
>
>
```

5.1 Formatos para Treinamentos e Predições

Para cada mês de previsão (agosto a dezembro de 2020), serão treinados modelos em um *dataset* com a seguinte composição:

- *Outcome* equivalente aos TPV's de julho 2020;
- A projeção, via regressão exponencial, para julho de 2020;
- TPV's mensais, iniciando no mês $M - x$, onde x é o nº de meses à frente que se deseja prever (para agosto, $x = 1$, setembro, $x = 2$, e assim por diante), sendo $M - 0$ equivalente a julho 2020;
- Dados cadastrais selecionados (porte, ticket e tipo de documento, além do TPV estimado).

O modelo treinado será aplicado para previsão em um *dataset* com o seguinte formato:

- A projeção, via regressão exponencial, para o mês em previsão;
- Os TPV's mensais, excluindo-se os x últimos, para manter o alcance máximo de 36 meses (a previsão para agosto 2020 não contará com o mês 37, para setembro 2020, com os meses 36 e 37, e assim por diante);
- Dados cadastrais selecionados (porte, ticket e tipo de documento, além do TPV estimado).

6 TREINAMENTOS E PREDIÇÕES

Para modelar os dados, foi escolhido um *ensemble* dos algoritmos `knn` e `glmnet` da `caret`. Suas predições seram combinadas usando o `gbm`. Os algoritmos do *ensemble* foram escolhidos por sua rapidez e bom desempenho geral. O algoritmo de combinação foi escolhido principalmente por sua robustez.

Os dados passarão pelas seguintes etapas antes de alimentar o *framework* de treinamento:

- A *feature* `jul20` receberá os dados das previsões para julho 2020 com os vários meses de diferença (até junho 2020 para a previsão de agosto - 1 mês de diferença, até maio 2020 para a previsão de setembro - 2 meses de diferença, e assim por diante).
- Serão selecionadas as colunas correspondentes ao mês de julho 2020 e aos meses anteriores que serão usados como preditores. A esse *dataset*, será adicionado o conjunto de dados cadastrais.

- Serão criadas variáveis binárias para os dados cadastrais.
- Serão selecionadas as instâncias pertencentes a cada *cluster* gerado anteriormente. Para esses dados, serão escolhidas a variáveis preditoras, usando a função `predSel`.
- Os dados serão, então, normalizados entre 0 e 1. Essa transformação não será aplicada ao *target* (TPV de julho 2020).

A junção dos modelos kNN e GLM se dará com o objetivo de maximizar o coeficiente de determinação R^2. Essa estratégia foi escolhida para que o *ensemble* explique da melhor forma possível os dados. Já o treinamento do GBM se dará visando a minimizar o MAE.

```
meses <- c("ago","set","out","nov","dez")
meses_ext <- c("agosto","setembro","outubro","novembro","dezembro")

mae <- matrix(NA,ncol = k,nrow = 5)

# computação em paralelo
Mycluster = makeCluster(detectCores()-2,
                        setup_strategy = "sequential")
registerDoParallel(Mycluster)

ini_time <-  Sys.time()

list_modelos <- list()
for (j in 1:5) {
  t0 <- Sys.time()

  cat("\nMês ");cat(j);cat("\n-------------\n")
  df_train_tpv$jul20 <- proj_tr[,paste0(meses[j],"20")]

  # Montagem do dataset de treino:

  if (j == 1) {
    df0 <- cbind(df_train_tpv[,-c(1,3:7)],
                 df_train_cad[,-1])
  }else{
    df0 <- cbind(df_train_tpv[,-c(1,3:7,9:(9+j-2))],
                 df_train_cad[,-1])
  }

  names(df0) <- c("proj.outcome",
                  "outcome",
                  paste0("M",j:36),
                  names(df_train_cad)[-1])

  df0 <- dummy_cols(df0,remove_first_dummy = T,
                    remove_selected_columns = T)
  rownames(df0) <- clientes %>% as.character

  mod_norm <- preProcess(df0[,-2],
                         method = "range",
                         rangeBounds = c(0,1))

  # Separação dos *clusters* e modelagem:
```

```
modelos <- list()
for (i in 1:k) {
  cat("Grupo ");cat(i);cat("\n")
  df1 <- df0[id_cluster[[i]] %>% as.character,]

  # feature selection
  pred_sel <- featSel(df1)

  # normalização
  df1 <- predict(mod_norm,df1)

  # transformação logarítmica
  df2 <- df1[,c("outcome",pred_sel)]

  modelo_list <- caretList(outcome ~ .,
                           df2,
                           metric="Rsquared",
                           trControl = trainControl(method = "repeatedcv",
                                                    repeats = 3,
                                                    number = 10,
                                                    summaryFunction = defaultSummary),
                           tuneList = list(
                             glm = caretModelSpec(method = "glmnet"),
                             knn = caretModelSpec(method = "knn",
                                                  tuneGrid = expand.grid(k = 9))
                           ))

  # ensembling ----
  modelo_ensemb <- caretStack(modelo_list,
                              method = "gbm",
                              tuneGrid = expand.grid(shrinkage=0.1,
                                                     interaction.depth=3,
                                                     n.trees=200,
                                                     n.minobsinnode=10),
                              metric = "MAE",
                              trControl = trainControl(method = "repeatedcv",
                                                       number = 10,
                                                       repeats = 3,
                                                       summaryFunction = defaultSummary))

  mae[j,i] <- modelo_ensemb$ens_model$results[,"MAE"] %>% min

  modelos[[i]] <- modelo_ensemb
  rm(df1,df2)
}

list_modelos[[meses[j]]] <- modelos

# Montagem do *dataset* de predição

df2 <- cbind(df_train_tpv[,c(1,j+2,8:(44-j))],
             df_train_cad[,-1])
```

```
  names(df2) <- c("id","proj.outcome",
                  paste0("M",j:36),
                  names(df_train_cad)[-1])
  df2 <- dummy_cols(df2,remove_first_dummy = T,
                    remove_selected_columns = T)

  # Predição

  pred_mes <- NULL
  for (i in 1:k) {
    df1 <- df2[df2$id %in% id_cluster[[i]],]
    pred <- predict(modelos[[i]],
                    predict(mod_norm,df1))
    pred_mes <- rbind(pred_mes,
                      cbind(df1$id,pred))
  }

  pred_mes <- pred_mes[order(pred_mes[,1]),]

  rm(df0,df1,df2)

  if (j == 1) {
    pred_full <- pred_mes
  }else{
    pred_full <- cbind(pred_full,pred_mes[,2])
  }

  rm(pred_mes)

  t1 <- Sys.time()
  cat("\nTempo de execução do mês: \n")
  print(t1 - t0)
}

colnames(pred_full) <- c("id",paste0("TPV ",meses_ext))

end_time <-  Sys.time()

# setup inicial de processamento
stopCluster(Mycluster)
registerDoSEQ()

cat("\n")
print(end_time - ini_time)
```

Vejamos como performaram os modelos durante os treinamentos:

```
colnames(mae) <- paste0("grupo_",1:7)
rownames(mae) <- meses
```

```
mae
```

```
>       grupo_1  grupo_2  grupo_3   grupo_4  grupo_5  grupo_6   grupo_7
>  ago 4776.868 2789.516 2101.210  5993.645 11005.65 3368.276  6996.382
>  set 6305.815 3001.903 2411.916  7823.537 15280.83 4113.436  8631.251
```

```
>  out 7442.680 3008.189 2644.003  8952.076 17658.49 4637.266  9644.928
>  nov 8718.893 3073.149 2694.708 10282.245 20774.24 5235.554 10754.357
>  dez 9315.832 3127.854 2795.522 10895.310 23790.14 5470.634 11055.326
```

Médias de MAE por grupo:

```
colMeans(mae)
```

```
>    grupo_1   grupo_2   grupo_3   grupo_4   grupo_5   grupo_6   grupo_7
>   7312.018  3000.122  2529.472  8789.363 17701.870  4565.033  9416.449
```

Fazendo uma média ponterada pelos tamanhos dos grupos, poderemos ter uma estimativa do MAE das previsões:

```
sum(sapply(id_cluster,
           length,
           simplify = T)*colMeans(mae))/length(clientes)
```

```
>  [1] 7539.949
```

```
df_train_tpv$ref_2020.07.31 %>%
  abs %>%
  mean
```

```
>  [1] 24628.92
```

Em relação ao valor da média absoluta dos TPV's de julho, o MAE esperado ficou em torno de 30%, considerando todos os 5 meses de previsão.

Médias de MAE por mês:

```
rowMeans(mae)
```

```
>       ago      set      out      nov      dez
>  5290.221 6795.527 7712.519 8790.449 9492.945
```

Naturalmente, quanto mais distante a predição, maior o erro envolvido.

(**EDA**) Observemos as distribuições e a sumarização das variações de TPV desde Julho:

```
TPV_07_12 <- data.frame(TPV.julho = df_train_tpv$ref_2020.07.31,
                        pred_full[,-1])

var_tpv <- (TPV_07_12[,2:6]-TPV_07_12[,1:5])
```

```
summary(var_tpv)
```

```
>     TPV.agosto          TPV.setembro        TPV.outubro
>   Min.   :-453919.7   Min.   :-149165.1   Min.   :-158561.0
>   1st Qu.:    878.2   1st Qu.:   -740.2   1st Qu.:  -1334.6
>   Median :   1952.9   Median :    528.0   Median :    452.9
>   Mean   :   2635.0   Mean   :    966.4   Mean   :   1042.7
>   3rd Qu.:   3719.0   3rd Qu.:   2348.2   3rd Qu.:   2999.9
>   Max.   : 164377.9   Max.   : 109413.5   Max.   : 149964.5
>    TPV.novembro        TPV.dezembro
>   Min.   :-200475.6   Min.   :-218835.1
>   1st Qu.:  -4321.1   1st Qu.:  -2855.8
>   Median :   -609.1   Median :   -106.0
>   Mean   :  -2112.7   Mean   :   -726.2
>   3rd Qu.:   1460.6   3rd Qu.:   2255.9
>   Max.   : 203938.7   Max.   : 193289.2
```

Vejamos como se comportaram os dados reais em um período de 6 meses:

```
TPV_02_07 <- data.frame(fevereiro = df_train_tpv$ref_2020.02.29,
                        marco = df_train_tpv$ref_2020.03.31,
                        abril = df_train_tpv$ref_2020.04.30,
                        maio = df_train_tpv$ref_2020.05.31,
                        junho = df_train_tpv$ref_2020.06.30,
                        julho = df_train_tpv$ref_2020.07.31)

var_tpv_real <- (TPV_02_07[,2:6]-TPV_02_07[,1:5])
```

```
summary(var_tpv_real)
```

```
>       marco              abril               maio              junho
>   Min.   :-6123536   Min.   :-17897391   Min.   :-6421012   Min.   :-2379177.9
>   1st Qu.:   -4205   1st Qu.:    -4430   1st Qu.:    -368   1st Qu.:   -1279.1
>   Median :    -862   Median :     -725   Median :    1329   Median :     571.2
>   Mean   :   -1988   Mean   :    -1976   Mean   :    3369   Mean   :    1756.8
>   3rd Qu.:    1390   3rd Qu.:     1755   3rd Qu.:    4982   3rd Qu.:    3456.4
>   Max.   : 8678956   Max.   :  2835843   Max.   : 1084752   Max.   :  830258.3
>       julho
>   Min.   :-854686.2
>   1st Qu.:   -371.7
>   Median :   1506.5
>   Mean   :   3067.9
>   3rd Qu.:   4951.4
>   Max.   : 536591.4
```

A presença de valores negativos de TPV torna sem sentido uma variação percentual, mas podemos verificar uma coerência dimensional entre as variações dos dois períodos.

7 VARIÁVEIS POSSIVELMENTE ÚTEIS NA PREVISÃO

- Fatores macroeconômicos, como inflação e taxa básica de juros: dado que o faturamento é profundamente impactado tanto pelo poder de compra da população quanto pela facilidade de acesso a crédito por parte do empresário, seria interessante adicionar preditores que descrevessem esse contexto.
- Desempenho de empresas do setor na Bolsa de Valores: por conta da maior abundância de informações históricas e da ampla teoria já produzida a respeito de previsão de séries temporais financeiras, seria interessante realizar previsões para os setores (englobando os diversos ativos de cada setor) e incorporar tais previsões no modelo aqui requerido.