SaeNer-Anexo 3.2: Estimación SAE-NER

-

Los siguientes códigos tienen como fin ejemplificar el uso de las funciones construidas para nuestro SAE-NER, no se proveen bases de datos para el mismo.

A continuación se muestra la capacidad de las funciones `ner_ebp_nG_CC`, `NER_boots` y `mse_ebp_ner_CC`, para obtener las estimaciones EPPCs, errores estándares sin condición de conformabilidad (CC) y errores estándares con CC respectivamente. Además, se presenta como se pueden realizar varias estimaciones de SAE-NER fácilmente.

[ ]:
```
library(data.table)
    setDTthreads(percent = 100)
library(stringr)
library(magrittr)

#Para paralelizacion
library(doSNOW)

#para lmm
library(lme4)
```

1 Códigos para SAE-NER

[ ]:
```
ner_ebp_n1=function(IN, OUT, y.var, dom, x.vars, z0){
  ###
  i = 1
  eq1 = c(); eq2 = c();
  for (x in x.vars){
    i = i + 1
    eq1 = paste(eq1 ,IN , "$", x, " + ", sep= "")
    eq2 = paste(eq2 , "beta[", i ,"]*", OUT , "$", x, " + ", sep= "")
  }
  eq1 = paste(IN,"$",y.var," ~ " , eq1 ,"(1|", dom,")" , sep = "")
  eq1 = as.formula(eq1)
  eq2 =  paste("~ I(",eq2 , "beta[1] )" , sep = "")
  eq2 = as.formula(eq2)
  ###
  lmm = lme4::lmer(eq1,data = get(IN) ,REML = TRUE)
  w1 = warnings()
  w1 = paste0(w1, collapse = "¬")
  gradient = attributes(lmm)$optinfo$derivs$gradient
  Hessian = attributes(lmm)$optinfo$derivs$Hessian
  w2 = attributes(lmm)$optinfo$warnings
  w2 = paste0(w2, collapse = "¬")
  opt_val = attributes(lmm)$optinfo$val

  beta<-lme4::fixef(lmm)
```

```
  list_doms = unique(get(IN)[[dom]])
  ud = data.table(list_doms, unname(unlist(ranef(lmm)))  )
  names(ud)=c(dom,"ud")
  var<-as.data.frame(VarCorr(lmm))
  sigmau2<-var$vcov[1]
  sigmae2<-var$vcov[2]
  ###
  y_est = data.table(as.numeric(model.frame(eq2)[,1]))
  y_est[,(dom)] = get(OUT)[[dom]]
  ###
  y_est=merge.data.table(y_est, ud, all=TRUE, by=(dom))
  names(y_est)[c(2:3)]=c("xbeta","ud")
  y_est[,mu_est:=xbeta+ud]
  ###
  get(IN)[, u := 1]
  nd = get(IN)[, sum(u), by= get(dom)][[2]]
  gammad=sigmau2*nd/(sigmau2*nd+sigmae2)
  vd_est=sigmau2*(1-gammad)+sigmae2
  vd_est=data.table(vd_est, list_doms)
  names(vd_est) = c('vd_est', dom)
  y_est=merge.data.table(y_est, vd_est, all=TRUE, by=(dom))
  ###
  y_est[,alpha_est :=vd_est^(-1/2)*(z0-(mu_est) )]
  ###
  y_est[,p_est_eb_j :=pnorm(alpha_est,mean=0,sd=1)]
  ###
  ebp = y_est[,mean(p_est_eb_j), by= eval(dom)]
  names(ebp) = c(dom,"ebp")
  w = list(w1 = w1, gradient = gradient , Hessian = Hessian , w2 =w2 , opt_val =opt_val )
  singular.fit = lmm  %>% isSingular
  AIC = lmm  %>% AIC(., k = 2)
  out = list(ebp, beta, sigmau2, sigmae2, w,singular.fit,AIC)
  names(out) = c("ebp", "beta", "sigmau2", "sigmae2", "w","singular.fit","AIC")
  return(out)
}
ner_ebp_n0=function(OUT_nd0, y.var, dom,
                    x.vars, beta, sigmau2, sigmae2, z0){
  ###
  eq2_N0 = c()
  i = 1
  for (x in x.vars){
    i = i +1
    eq2_N0 = paste(eq2_N0 , "beta[", i ,"]*", OUT_nd0 , "$", x, " + ", sep= "")
  }
  eq2_N0 =  paste("~ I(",eq2_N0 , "beta[1] )" , sep = "")
  eq2_N0 = as.formula(eq2_N0)
  ###
  y_est_N0 = data.table(as.numeric(model.frame(eq2_N0)[,1]))
  y_est_N0[,(dom)]= get(OUT_nd0)[[dom]]
  names(y_est_N0)[1] = "xb"
  ###
  vd_N0 = sigmau2+sigmae2
  ###
  y_est_N0[,alpha_est :=vd_N0^(-1/2)*(z0-xb )]
  ###
  y_est_N0[,p_est_eb_j :=pnorm(alpha_est,mean=0,sd=1)]
  ###
  ebp_N0 = y_est_N0[,mean(p_est_eb_j), by= eval(dom)]
  names(ebp_N0) = c(dom,"ebp")
  out = list("ebp_N0" = ebp_N0, "vd_N0" = vd_N0)
  return(out)
}
ner_ebp_nG = function(IN, OUT_f, y.var, dom, x.vars, z0){

  w_noIN = which(get(OUT_f)[[dom]] %in% unique(get(IN)[[dom]]) )
  if (length(w_noIN)>0){
```

```
    out_n1 <<-get(OUT_f)[w_noIN,]
    out_n0 <<-get(OUT_f)[-w_noIN,]

    temp1 <<- ner_ebp_n1(IN = IN, OUT = "out_n1",
                         y.var = y.var, dom = dom,
                         x.vars = x.vars, z0 = z0)

    temp0 <<- ner_ebp_n0(OUT_nd0 = "out_n0",
                         y.var = y.var, dom = dom,
                         x.vars = x.vars,
                         beta=temp1[[2]], sigmau2=temp1[[3]],
                         sigmae2=temp1[[4]], z0=-2)

    ebp_f = rbind(temp1[["ebp"]],temp0[["ebp_N0"]]) %>% data.table
    ebp_f[,n_d := 0]
    ebp_f[1:dim( temp1[["ebp"]] )[1],n_d := 1]
    ebp_f = ebp_f[order(get(dom)),]
    temp1[["ebp"]] = ebp_f
    rm(out_n1, out_n0, temp1,temp0, envir = .GlobalEnv)
    return(temp1)
  }


  if (length(w_noIN)==0){
    temp1 = ner_ebp_n1(IN = IN, OUT = OUT_f,
                       y.var = y.var, dom = dom,
                       x.vars = x.vars, z0 = z0)
    return(temp1)
  }


  if (length(get(OUT_f)[[dom]]) <length(unique(get(IN)[[dom]]))){
    print("Sample cannot have more domains than population")
  }
}
NER_boots = function(IN,OUT,X.var,Y.var,
                     domain,poverty_line,B_iter.boots) {
  ##NER
  NER_res = ner_ebp_nG(IN = IN, OUT_f = OUT,
                       y.var = Y.var, dom = domain,
                       x.vars = X.var, z0 = poverty_line)

  sigmau2 = NER_res[["sigmau2"]]
  sigmae2 = NER_res[["sigmae2"]]
  beta = NER_res[["beta"]]

  #Ecuaciones para obtener xb en IN y OUT
  get(IN)[, intercept:= 1]
  get(OUT)[, intercept:= 1]
  names_b = names(beta)
  names_b[1] = paste0(IN, "$intercept")
  names_b = paste0(paste0("beta[", 1:length(names_b), "]","*") , names_b)
  eq1 = paste(names_b, collapse = " + ")
  eq2 = gsub(IN, OUT, eq1)
  eq1 = paste0("~I(", eq1, ")") %>% as.formula
  eq2 = paste0("~I(", eq2, ")") %>% as.formula
  ## Se obtienen xb
  y_est_sample = data.table(as.numeric(model.frame(eq1)[,1]))
  y_est_census = data.table(as.numeric(model.frame(eq2)[,1]))

  y_est_sample[, get("domain") := get(IN)[[domain]]]
  y_est_census[, get("domain") := get(OUT)[[domain]]]

  names(y_est_sample)[1] = "y_est"
  names(y_est_census)[1] = "y_est"

  ##Total de dominios
```

```
Nd = length(unique(get(OUT)[,..domain][[1]]))

##Tamaño de censo más encuesta
nN = dim(get(IN))[1] + dim(get(OUT))[1]

## pre-alocación de objetos (ahorra memoria en relación a una list)
temp_boots_1 = matrix(data=0,nrow = Nd, ncol = B_iter.boots)
temp_boots_2 = matrix(data=0,nrow = Nd, ncol = B_iter.boots)
temp_boots_w = matrix(data="",ncol = 6, nrow = B_iter.boots)
colnames(temp_boots_w) = c("w1", "gradient", "Hessian" , "w2", "opt_val", "iteración")
#temp_boots_1 = list()
#temp_boots_2 = list()
#temp_boots_w = list()
singular.fit.m = matrix(data=F,nrow = B_iter.boots, ncol = 1)
purrr_errors = list()

#u_d y e_dj vacias
u_d_asterisc_dt =  rep(0,Nd) %>% data.table
u_d_asterisc_dt[, get("domain") := unique(get(OUT)[[domain]]) ]
names(u_d_asterisc_dt)[1] = "u_d_asterisc"
e_dj_asterisc_dt = rep(0,nN) %>% data.table
e_dj_asterisc_dt[1:dim(get(IN))[1], i :=1]
e_dj_asterisc_dt[(dim(get(IN))[1] +1):nN, i :=2]
names(e_dj_asterisc_dt)[1] = "e_dj_asterisc"

##Bootstrap Loop
for (b.i in 1:B_iter.boots){
  set.seed(b.i)
  #print(b.i)
  ##
  u_d_asterisc_dt[,u_d_asterisc := rnorm(Nd, mean = 0, sd = sigmau2^.5)]
  e_dj_asterisc_dt[,e_dj_asterisc := rnorm(nN, mean = 0, sd = sigmae2^.5)]
  ##
  suppressWarnings(y_est_sample[, u_d_asterisc := NULL])
  suppressWarnings(y_est_census[, u_d_asterisc := NULL])
  y_est_sample=merge(y_est_sample, u_d_asterisc_dt, all=FALSE, all.x=T, all.y=F, by=(domain))
  y_est_census=merge(y_est_census, u_d_asterisc_dt, all=TRUE, by=(domain))

  y_est_sample[,e_dj_asterisc := e_dj_asterisc_dt[i == 1][["e_dj_asterisc"]]]
  y_est_census[,e_dj_asterisc := e_dj_asterisc_dt[i == 2][["e_dj_asterisc"]]]

  y_est_sample[ , xi := y_est + e_dj_asterisc + u_d_asterisc]
  y_est_census[ , xi := y_est + e_dj_asterisc + u_d_asterisc]

  ##
  y_est_census[, descro_asterisc := as.numeric(y_est_census[["xi"]] < poverty_line)]
  f_d_asterisc = y_est_census[,mean(descro_asterisc), by=get(domain)][[2]]
  ##
  get(IN)[, xi := y_est_sample[["xi"]]]
  ##
  ner.a = function() ner_ebp_nG(IN = IN, OUT_f = OUT,
                                y.var = "xi", dom = domain,
                                x.vars = X.var, z0 = -2)
  ner.a.s = purrr::safely(ner.a)
  tmp_NER_saf = ner.a.s()

  temp_NER_res =tmp_NER_saf[[1]]
  f_d_EBP_asterisc= temp_NER_res[["ebp"]][[2]]

  ##
  temp_boots_w[b.i,] = c(unlist(temp_NER_res$w),b.i)
  ##
  temp_boots_1[,b.i] = f_d_EBP_asterisc
  temp_boots_2[,b.i] = f_d_asterisc
  purrr_errors[[b.i]] = tmp_NER_saf[-1]
  singular.fit.m[b.i,1] = tmp_NER_saf[[1]][["singular.fit"]]
}
```

```
  #temp_boots_1  %<>% do.call(cbind,.)
  #temp_boots_2  %<>% do.call(cbind,.)
  ##Cálculo MSE Bootstrap
  mse_boots = (rowSums((temp_boots_1 - temp_boots_2)^2))/B_iter.boots

  ###Resultados
  res=list(mse_boots,temp_boots_1, temp_boots_2,temp_boots_w,purrr_errors,singular.fit.m )
  names(res) = c("mse_boots", "temp_boots_1", "temp_boots_2","temp_boots_w","purrr_errors","singular.fit.m")
  #Limpieza
  get(IN)[, intercept:= NULL]
  get(OUT)[, intercept:= NULL]
  ##
  return(res)
}

ner_ebp_nG_CC = function(IN, OUT_f, y.var, dom, x.vars, z0,BC_dt){
  temp1 =
    ner_ebp_nG(IN=IN, OUT_f= OUT_f, y.var=y.var, dom = dom,
               x.vars=x.vars, z0=z0)
  #Condición Benchmark o de Conformidad
  #Variable de estimación para condición benchmark debe llamarse y_CC
  #Variable para identificar dominio debe llamarse dom
  #Variable para identificar dominio superior confiable debe llamarse id_CC
  ebp_B = temp1$ebp
  BC_dt = merge(BC_dt, ebp_B, by = dom, all = T)

  delta_CC = merge(BC_dt[, weighted.mean(ebp, pob_tot), by = "id_CC"]  %>% setnames("V1", "ebp_S"),
                   BC_dt[, mean(y_CC), by = "id_CC"] %>% setnames("V1", "y_CC"),
                   by = "id_CC", all = T)

  delta_CC[, delta:= y_CC/ebp_S]
  delta_CC = delta_CC[, .(id_CC, delta)]
  ebp_CC = merge(BC_dt, delta_CC, by = "id_CC", all = T)
  ebp_CC[, ebp_CC := ebp*delta]
  ebps = list("ebp_noCC" = temp1, "ebp_CC" =  ebp_CC)
  return(ebps)
}

NER_boots_CC = function(IN,OUT,X.var,Y.var,
                        domain,poverty_line,B_iter.boots,BC_dt) {
  ##NER
  NER_res = ner_ebp_nG(IN = IN, OUT_f = OUT,
                       y.var = Y.var, dom = domain,
                       x.vars = X.var, z0 = poverty_line)

  sigmau2 = NER_res[["sigmau2"]]
  sigmae2 = NER_res[["sigmae2"]]
  beta = NER_res[["beta"]]

  #Ecuaciones para obtener xb en IN y OUT
  get(IN)[, intercept:= 1]
  get(OUT)[, intercept:= 1]
  names_b = names(beta)
  names_b[1] = paste0(IN, "$intercept")
  names_b = paste0(paste0("beta[", 1:length(names_b), "]","*") , names_b)
  eq1 = paste(names_b, collapse = " + ")
  eq2 = gsub(IN, OUT, eq1)
  eq1 = paste0("~I(", eq1, ")") %>% as.formula
  eq2 = paste0("~I(", eq2, ")") %>% as.formula
  ## Se obtienen xb
  y_est_sample = data.table(as.numeric(model.frame(eq1)[,1]))
  y_est_census = data.table(as.numeric(model.frame(eq2)[,1]))

  y_est_sample[, get("domain") := get(IN)[[domain]]]
  y_est_census[, get("domain") := get(OUT)[[domain]]]

  names(y_est_sample)[1] = "y_est"
```

```
names(y_est_census)[1] = "y_est"

##Total de dominios
Nd = length(unique(get(OUT)[,..domain][[1]]))

##Tamaño de censo más encuesta
nN = dim(get(IN))[1] + dim(get(OUT))[1]

## pre-alocación de objetos (ahorra memoria en relación a una list)
temp_boots_1 = matrix(data=0,nrow = Nd, ncol = B_iter.boots)
temp_boots_2 = matrix(data=0,nrow = Nd, ncol = B_iter.boots)
temp_boots_w = matrix(data="",ncol = 6, nrow = B_iter.boots)
colnames(temp_boots_w) = c("w1", "gradient", "Hessian" , "w2", "opt_val", "iteración")
singular.fit.m = matrix(data=F,nrow = B_iter.boots, ncol = 1)

#u_d y e_dj vacias
u_d_asterisc_dt =  rep(0,Nd) %>% data.table
u_d_asterisc_dt[, get("domain") := unique(get(OUT)[[domain]]) ]
names(u_d_asterisc_dt)[1] = "u_d_asterisc"
e_dj_asterisc_dt = rep(0,nN) %>% data.table
e_dj_asterisc_dt[1:dim(get(IN))[1], i :=1]
e_dj_asterisc_dt[(dim(get(IN))[1] +1):nN, i :=2]
names(e_dj_asterisc_dt)[1] = "e_dj_asterisc"

##Bootstrap Loop
for (b.i in 1:B_iter.boots){
  set.seed(b.i)
  #print(b.i)
  ##
  u_d_asterisc_dt[,u_d_asterisc := rnorm(Nd, mean = 0, sd = sigmau2^.5)]
  e_dj_asterisc_dt[,e_dj_asterisc := rnorm(nN, mean = 0, sd = sigmae2^.5)]
  ##
  suppressWarnings(y_est_sample[, u_d_asterisc := NULL])
  suppressWarnings(y_est_census[, u_d_asterisc := NULL])
  y_est_sample=merge(y_est_sample, u_d_asterisc_dt, all=FALSE, all.x=T, all.y=F, by=(domain))
  y_est_census=merge(y_est_census, u_d_asterisc_dt, all=TRUE, by=(domain))

  y_est_sample[,e_dj_asterisc := e_dj_asterisc_dt[i == 1][["e_dj_asterisc"]]]
  y_est_census[,e_dj_asterisc := e_dj_asterisc_dt[i == 2][["e_dj_asterisc"]]]

  y_est_sample[ , xi := y_est + e_dj_asterisc + u_d_asterisc]
  y_est_census[ , xi := y_est + e_dj_asterisc + u_d_asterisc]

  ##
  y_est_census[, descro_asterisc := as.numeric(y_est_census[["xi"]] < poverty_line)]
  f_d_asterisc = y_est_census[,mean(descro_asterisc), by=get(domain)][[2]]

  get(IN)[, xi := y_est_sample[["xi"]]]
  ###
  temp_NER_res = ner_ebp_nG_CC(IN = IN, OUT_f = OUT,
                               y.var = "xi", dom = domain,
                               x.vars = X.var, z0 = poverty_line,
                               BC_dt =BC_dt)
  f_d_EBP_asterisc= temp_NER_res[["ebp_CC"]][, ebp_CC]
  temp_boots_w[b.i,] = c(unlist(temp_NER_res[["ebp_noCC"]]$w),b.i)

  temp_boots_1[,b.i] = f_d_EBP_asterisc
  temp_boots_2[,b.i] = f_d_asterisc
  singular.fit.m[b.i,1] = temp_NER_res[["ebp_noCC"]][["singular.fit"]]
}

##Cálculo MSE Bootstrap
mse_boots = (rowSums((temp_boots_1 - temp_boots_2)^2))/B_iter.boots
#%>%    data.table
###Resultados
res=list(mse_boots,temp_boots_1, temp_boots_2,temp_boots_w,singular.fit.m )
names(res) = c("mse_boots", "temp_boots_1", "temp_boots_2","temp_boots_w","singular.fit.m")
```

```
  #Limpieza
  get(IN)[, intercept:= NULL]
  get(OUT)[, intercept:= NULL]
  ##
  return(res)
}
###

to_formula_lmm<-function(y,X,dom){
  .X = paste(X, collapse = '+')
  .f = paste0(y,'~',.X, ' + (1|', dom, ')')
  .f = .f %>% as.formula()
  return(.f)
}

## DV
##Función glmm
glmmDv = function(eqlmer,IN){
  #,g_control
  #, envir = .GlobalEnv
  glmmf<-function() lme4::lmer(formula= eqlmer,data=get(IN))
  #purrr safely (?) útil
  glmmPurrrS = purrr::safely(.f = glmmf)
  glmmR = glmmPurrrS()

  w1 = warnings()
  w1 = paste0(w1, collapse = "¬")
  ##!glmm resultados limpios
  glmm = glmmR$result
  #resultados
  AIC_glmm = glmm %>% AIC
  AIC2_glmm = glmm %>% AIC(k=2)
  BIC_glmm = glmm %>% BIC

  w2 = attributes(glmm)$optinfo$warnings
  w2 = paste0(w2, collapse = "¬")
  w3 = attributes(glmm)$optinfo$conv$lme4$messages
  w3 = paste0(w3, collapse = "¬")
  singular = lme4::isSingular(glmm)
  singular = ifelse(singular,"singular", "")

  #Información purrr
  if (is.null(glmmR[["error"]])) glmmR[["error"]] = ""
  res2 = glmmR[["error"]]

  #Resumen resultados
  res1 = data.table('AIC'= AIC_glmm,
                    'AIC2'= AIC2_glmm,
                    'BIC'= BIC_glmm,
                    'w1'=w1, 'w2' = w2, 'w3'=w3,
                    'singular' = singular,
                    'purrr_e' = res2  %>% unlist  %>% paste(collapse=""))

  res3 = res1[, .( w2, w3,purrr_e,singular)]  %>% paste(collapse="")

  res = list('res1' = res1, 'res2' = res2,'res3' = res3)
  gc()
  return(res)
}
## Se estiman todos los glmm
glmm_all_txs = function(IN, y.var, domain,x.vars_t.x, parallel = F ){
  all_Gl.ob = ls(.GlobalEnv, all.names = T)
  all_Gl.ob %<>%   .[sapply(. ,function(x) is.function(get(x))) ]
  all_Gl.ob = c(all_Gl.ob, IN)
  for(tE in all_Gl.ob) assign(tE, get(tE, envir = .GlobalEnv))
```

```
  l.tx = dim(x.vars_t.x)[2]

  if (parallel == T) "%bla%" <-  `%dopar%`
  if (parallel == F) "%bla%" <-  `%do%`

  toExp = ls(environment(), all.names = T)
  toExp  %<>%  str_subset("domain|formjc|glmmDv|IN|x.vars_t.x|y.var", negate = T)



  AICS.x = foreach(t.x = 1:l.tx, .combine=rbind, .packages = c("data.table","magrittr"),
                   .export=toExp, .verbose = F) %bla%{
                     x.tx = x.vars_t.x[,t.x]
                     eqgl_tx = to_formula_lmm(y = y.var,X = x.tx,
                                              dom = domain)
                     glmm_txA = glmmDv(eqlmer = eqgl_tx,IN = IN)
                     glmm_tx = glmm_txA$res1
                     Xs = paste(x.tx, collapse = ",")
                     AICS.tx = data.table(id = t.x, glmm_tx, all_errors = glmm_txA$res3,
                                          Xs = Xs)
                     gc()
                     return(AICS.tx)
                   }
  gc()
  return(AICS.x)
}




step_AIC_glmm.forw.dv = function(y.var,x.vars_t0,domain,IN,parallel){
  step_AIC.DV = list()

  AIC_t0 = Inf
  AIC_MIN.tx = Inf
  errors_tx = ""

  in_X.t0 = c()
  x.var_tx = x.vars_t0
  ###While

  i = 0
  while ( (AIC_t0 >= AIC_MIN.tx) & (errors_tx == "")    ){
    #while (i <=4){
    t0 = Sys.time()
    #for (atx in 1:3){
    gc()
    i = i + 1
    #Se generan todas las comb
    x.var_tx_all = x.var_tx  %>% combn(., m = 1)
    l_all = dim(x.var_tx_all)[2]
    if(is.null(in_X.t0)==F) x.var_tx_all = rbind(x.var_tx_all,matrix(data = in_X.t0, nrow = length(in_X.
↪t0), ncol =l_all ))
    #Se obtienen estadisticos de cada comb

    AIC.dt.x = glmm_all_txs(IN = IN, y.var = y.var, domain = domain,
                            x.vars_t.x = x.var_tx_all,parallel = parallel)
    AIC.dt.x[,"Xs_in"] = AIC.dt.x[, Xs] %>% lapply(., function(x) x.var_tx[x.var_tx %in%␣
↪(unlist(str_split(x,","))) ==T] %>% data.table  ) %>% rbindlist

    tot_errors=AIC.dt.x[all_errors !=""] %>% dim  %>% .[1]
    tot_comb=AIC.dt.x %>% dim  %>% .[1]
    ##Dos escenarios:
    #1) No todas las comb tienen errores, se extrae info del mejor AIC
```

```
    if (tot_comb>tot_errors){
      AIC_MIN.tx = AIC.dt.x[all_errors ==""][AIC_glmm == min(AIC_glmm) ,AIC_glmm]
      id_tx = AIC.dt.x[all_errors ==""][AIC_glmm == min(AIC_glmm),id]

      errors_tx = AIC.dt.x[all_errors ==""][AIC_glmm == min(AIC_glmm),all_errors]
      in_X.tx = AIC.dt.x[all_errors ==""][AIC_glmm == min(AIC_glmm),Xs_in]

    }
    #2) Todas las comb tienen errores, se extrae info del mejor AIC (con error)
    if (tot_comb==tot_errors){
      AIC_MIN.tx = AIC.dt.x[AIC_glmm == min(AIC_glmm),AIC_glmm]
      id_tx = AIC.dt.x[AIC_glmm == min(AIC_glmm),id]

      errors_tx = AIC.dt.x[AIC_glmm == min(AIC_glmm),all_errors]
      in_X.tx = AIC.dt.x[AIC_glmm == min(AIC_glmm),Xs_in]

    }
    x.var_tx = x.var_tx[(x.var_tx %in% x.var_tx_all[, id_tx])==F]
    in_X.t0 = c(in_X.t0, in_X.tx)
    #Resumen resultados de iteración y print
    gc()
    t1 = Sys.time()
    t.total = t1-t0
    print(rep("*",60) %>% paste(collapse = ""))
    print(paste0("TotVars: ",l_all,"- AIC:",round(AIC_MIN.tx,3)," id_best: ",id_tx, " inVar: ",in_X.tx,␣
↪"--- Error: " ,errors_tx))
    print(t.total)
    res = data.table(nIter = i, AIC_min = AIC_MIN.tx, idminXs = id_tx, lMinXs =l_all,
                     errors = errors_tx, Xs = x.var_tx_all[, id_tx]  %>% paste( collapse = ","),in_X.tx =␣
↪in_X.tx,
                     t.total = t.total)

    #Se exporta resumen final de datos
    step_AIC.DV[[i]] = res
    #Para conocer t-1
    if (i>1) AIC_t0 = step_AIC.DV[[i-1]][,AIC_min]


  }
  step_AIC.DV.dt = do.call(rbind,step_AIC.DV)
  return(step_AIC.DV.dt)
}

#fun_in could be: 'AIC', 'BIC'
step_fun_glmm.forw.dv = function(y.var,x.vars_t0,domain,IN,parallel,fun_in){
  step_AIC.DV = list()

  AIC_t0 = Inf
  AIC_MIN.tx = Inf
  errors_tx = ""

  in_X.t0 = c()
  x.var_tx = x.vars_t0
  ###While

  i = 0
  while ( (AIC_t0 >= AIC_MIN.tx) & (errors_tx == "")    ){
    #while (i <=4){
    t0 = Sys.time()
    #for (atx in 1:3){
    gc()
    i = i + 1
    #Se generan todas las comb
    x.var_tx_all = x.var_tx  %>% combn(., m = 1)
    l_all = dim(x.var_tx_all)[2]
    if(is.null(in_X.t0)==F) x.var_tx_all = rbind(x.var_tx_all,matrix(data = in_X.t0, nrow = length(in_X.
↪t0), ncol =l_all ))
```

```
    #Se obtienen estadisticos de cada comb

    AIC.dt.x = glmm_all_txs(IN = IN, y.var = y.var, domain = domain,
                            x.vars_t.x = x.var_tx_all,parallel = parallel)
    AIC.dt.x[,"Xs_in"] = AIC.dt.x[, Xs] %>% lapply(., function(x) x.var_tx[x.var_tx %in%
↪(unlist(str_split(x,","))) ==T] %>% data.table  ) %>% rbindlist

    tot_errors=AIC.dt.x[all_errors !=""] %>% dim  %>% .[1]
    tot_comb=AIC.dt.x %>% dim  %>% .[1]
    ##Dos escenarios:
    #1) No todas las comb tienen errores, se extrae info del mejor AIC
    if (tot_comb>tot_errors){
      AIC_MIN.tx = AIC.dt.x[all_errors ==""][get(fun_in) == min(get(fun_in)) ,get(fun_in)]
      id_tx = AIC.dt.x[all_errors ==""][get(fun_in) == min(get(fun_in)),id]

      errors_tx = AIC.dt.x[all_errors ==""][get(fun_in) == min(get(fun_in)),all_errors]
      in_X.tx = AIC.dt.x[all_errors ==""][get(fun_in) == min(get(fun_in)),Xs_in]

    }
    #2) Todas las comb tienen errores, se extrae info del mejor AIC (con error)
    if (tot_comb==tot_errors){
      AIC_MIN.tx = AIC.dt.x[get(fun_in) == min(get(fun_in)),get(fun_in)]
      id_tx = AIC.dt.x[get(fun_in) == min(get(fun_in)),id]

      errors_tx = AIC.dt.x[get(fun_in) == min(get(fun_in)),all_errors]
      in_X.tx = AIC.dt.x[get(fun_in) == min(get(fun_in)),Xs_in]

    }
    x.var_tx = x.var_tx[(x.var_tx %in% x.var_tx_all[, id_tx])==F]
    in_X.t0 = c(in_X.t0, in_X.tx)
    #Resumen resultados de iteración y print
    gc()
    t1 = Sys.time()
    t.total = t1-t0
    print(rep("*",60) %>% paste(collapse = ""))
    print(paste0("TotVars: ",l_all,"- AIC:",round(AIC_MIN.tx,3)," id_best: ",id_tx, " inVar: ",in_X.tx,
↪"--- Error: " ,errors_tx))
    print(t.total)
    res = data.table(nIter = i, AIC_min = AIC_MIN.tx, idminXs = id_tx, lMinXs =l_all,
                     errors = errors_tx, Xs = x.var_tx_all[, id_tx]  %>% paste( collapse = ","),in_X.tx =
↪in_X.tx,
                     t.total = t.total)

    #Se exporta resumen final de datos
    step_AIC.DV[[i]] = res
    #Para conocer t-1
    if (i>1) AIC_t0 = step_AIC.DV[[i-1]][,AIC_min]


  }
  step_AIC.DV.dt = do.call(rbind,step_AIC.DV)
  return(step_AIC.DV.dt)
}


to_eq_SaeNer = function(IN, OUT, domain,Xs,y.var){
  i = 1
  eq1 = c(); eq2 = c();
  for (x in Xs){
    i = i + 1
    eq1 = paste(eq1 ,IN , "$", x, " + ", sep= "")
    eq2 = paste(eq2 , "beta[", i ,"]*", OUT , "$", x, " + ", sep= "")
  }
  eq1 = paste(IN,"$",y.var," ~ " , eq1 ,"(1|", domain,")" , sep = "")
  eq1 = as.formula(eq1)
  eq2 =  paste("~ I(",eq2 , "beta[1] )" , sep = "")
  eq2 = as.formula(eq2)
```

```
  return(list(eq1,eq2))
}
y_hat_lmm = function(IN, domain, OUT,lmm,eq2){
  beta=lme4::fixef(lmm)
  list_doms = unique(get(IN)[[domain]])
  ud = data.table(list_doms, unname(unlist(ranef(lmm)))  )
  names(ud)=c(domain,"ud")
  y_est = data.table(as.numeric(model.frame(eq2)[,1]))
  y_est[,(domain)] = get(OUT)[[domain]]
  y_est=merge.data.table(y_est, ud, all=TRUE, by=(domain))
  names(y_est)[c(2:3)]=c("xbeta","ud")
  y_est[,mu_est:=xbeta+ud]
  return(y_est)
}
```

2 SAE-NER para DCI nivel Cantón para ECV-2014

2.1 Se cargan bases

[ ]:
```
#Base de datos muestral
dat = readRDS("dat.rds")
setDT(dat)
#Base de datos censal
aux0 = readRDS("aux0.rds")
setDT(aux0)
#Objeto con las variables seleccionadas para SAE-NER, puede ser un solo conjunto
# o varios
all.Xs.fit = readRDS("Xs_SaeNer_aENET_ecv_all.rds")
```

2.2 Ejemplo de objeto con información necesaria para CC (Condición de Conformidad)

[ ]:
```
# `EstiDirecDescro_prov_f` objeto para CC
tot_aux_CC = aux0[,.N,by = get(domain)]
names(tot_aux_CC) = c("dom", "pob_tot")
tot_aux_CC[, id_CC:=substr(dom, 1,2)]

dat[, prov:= substr(get(domain),1,2)]

EstiDirecDescro_prov_f = NULL
EstiDirecDescro_prov_f =
  dat[, weighted.mean(dcronica,fexp), by = list(id_CC) ]
EstiDirecDescro_prov_f  %>% setnames(c("id_CC", "y_CC"))
EstiDirecDescro_prov_f = merge(EstiDirecDescro_prov_f,tot_aux_CC,
                               by = "id_CC", all = T)
EstiDirecDescro_prov_f %>% setnames('dom','id_dominio')

BC_dt = EstiDirecDescro_prov_f
```

3 Ejemplo uso funciones SAE-NER

[ ]:
```
#En character, nombre de base muestral:
IN = "dat"
#En character, nombre de base censal:
OUT = "aux0"
#En character, nombre de variable con identificador de cada "dominio":
domain = "id_dominio"
#En character, nombre de variable con y para regresión LMM:
Y.var = "y"
#En vector de "characters", nombre de variables Xs para regresión LMM:
X.var = all.Xs.fit[1]
```

```
    #Asumo que el objeto all.Xs.fit, tiene varios grupos de variables
    #Xs para LMM de SAE-NER. Caso contrario: X.var = all.Xs.fit
#Linea de pobreza:
poverty_line = -2
    #Para el caso de desnutrición crónica donde y es el Z-score
#data.table con información para Condición de Conformabilidad:
EstiDirecDescro_prov_f
#Total de iteraciones Bootstrap deseadas para cálculo de MSE de EPPCs
B_iter.boots = 300
#Nombre de variable cuyo nivel de desagregación se espera igualar con Conformidad
id_CC = "id_CC"

##
#Código para obtener estimaciones de EPPCs
EPPCs = ner_ebp_nG_CC(IN = IN, OUT_f = OUT,
                        y.var = Y.var, dom = domain,
                        x.vars = X.var, z0 = poverty_line,
                        BC_dt =EstiDirecDescro_prov_f)
#Código para obtener MSE de EPPs
mse_ebp_ner = NER_boots(IN = IN, OUT= OUT,
                       X.var = X.var,
                       Y.var = Y.var,
                       domain = domain,
                       poverty_line=poverty_line,
                       B_iter.boots=B_iter.boots)
#Código para obtener MSE de EPPCs
mse_ebp_ner_CC = NER_boots_CC(IN = IN, OUT= OUT,
                             X.var = X.var, Y.var = Y.var,
                             domain = domain,
                             poverty_line=poverty_line,
                             B_iter.boots=B_iter.boots,
                             BC_dt =EstiDirecDescro_prov_f)
```

4 Bucle para varios SAE-NER

[ ]:
```
all_sae_ner_l = list()
invisible(gc())
for (.i in 1:length(all.Xs.fit_noMult_l)){
    ebpXsLog_CC = NULL
    mse_ebp_ner=NULL
    mse_ebp_ner_CC=NULL

    X.var = all.Xs.fit_noMult_l[[.i]]
    aux0.red <<- aux0[, mget(c(X.var,id_CC,domain) )]

    invisible(gc())

    ##
    #Código para obtener estimaciones de EPPCs
    EPPCs = ner_ebp_nG_CC(IN = IN, OUT_f = OUT,
                             y.var = Y.var, dom = domain,
                             x.vars = X.var, z0 = poverty_line,
                             BC_dt = EstiDirecDescro_prov_f)
    #Código para obtener MSE de EPPs
    mse_ebp_ner = NER_boots(IN = IN, OUT= OUT,
                           X.var = X.var,
                           Y.var = Y.var,
                           domain = domain,
                           poverty_line=poverty_line,
                           B_iter.boots=B_iter.boots)
    #Código para obtener MSE de EPPCs
    mse_ebp_ner_CC = NER_boots_CC(IN = IN, OUT= OUT,
                                 X.var = X.var, Y.var = Y.var,
                                 domain = domain,
                                 poverty_line=poverty_line,
```

```
                                B_iter.boots=B_iter.boots,
                                BC_dt =EstiDirecDescro_prov_f)


    all_sae_ner_l[[.i]] = list("ebpXsLog_CC" = ebpXsLog_CC,
                               "mse_ebp_ner" = mse_ebp_ner,
                               "mse_ebp_ner_CC" = mse_ebp_ner_CC)
    invisible(gc())

}
```