SaeNer-Anexo 3.1: Selección de Variables

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 el uso de la `aENET_idealXs` para obtener variables sugeridas por Ada-ENET dado cualquier valor de α.

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

#Para ada enet
library(glmnet)
#Para paralelizacion
library(doSNOW)
```

1 Código para Ada-ENET

[ ]:
```
### Adaptative LASSO o ENET
# Posible families: c("gaussian", "binomial", "poisson", "multinomial","cox", "mgaussian")
# nlambda total de lambdas a probar
# alpha = 1 = LASSO
# Si n.f = 10 la bdd se parte en 10 pedazos, el décimo pedazo es la base test, se realiza el proceso 10␣
↪veces
# con seed = NULL se trabaja sin semilla
# n.i = total iteraciones sobre folds
# Si se require usar como función de pérdida diferente al MSE (menos es mejor), por ejemplo AUC (más es␣
↪mejor), se debe
# multiplicar por -1 la variable cvm en los data.tables respectivos


aENET_idealXs = function(X,y,n.f = 10, constant = T, parallel = F, alpha = 1,
                         nlambda = 100, family  = "gaussian", seed = NULL, n.i = 100){
  ### Seed or not seed?
  set.seed(seed)

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

  ### first step, pure ENET
  lasso1.s = foreach (t.x = 1:n.i, .combine=cbind , .packages =
                        c("data.table","magrittr")
  ) %bla% {
    if (is.null(seed)==F) seed.x = seed+t.x
```

```
  if (is.null(seed)==T) seed.x = NULL

  set.seed(seed.x)
  lasso1.s.x=glmnet::cv.glmnet(x = X, y = y,
                               family = family,
                               type.measure = "mse",
                               nfold = n.f,
                               alpha = alpha,
                               keep = TRUE,
                               nlambda = nlambda,
                               intercept = constant,
                               parallel = F)

  cvMs.x = data.table(cvm = lasso1.s.x$cvm,
                      lambda = lasso1.s.x$lambda)  %>%
    setnames(c("cvm","lambda"),
             paste0(c("cvm","lambda"),t.x))

  gc()
  return(cvMs.x)
}
n.lasso1.s = names(lasso1.s)
n.lasso1.s.c = n.lasso1.s[grepl("cvm", n.lasso1.s)]
cvms.s = data.table(lambda = lasso1.s[, lambda1],
                    cvm.m = lasso1.s[, mget(n.lasso1.s.c)] %>% rowMeans()
)
Best_Lambda = cvms.s[order(cvm.m),lambda][1]
if (is.null(seed)==F) seed.x = seed+1
if (is.null(seed)==T) seed.x = NULL

set.seed(seed.x)
lasso1.s.1=glmnet::cv.glmnet(x = X, y = y,
                             family = family,
                             type.measure = "mse",
                             nfold = n.f,
                             alpha = alpha,
                             keep = TRUE,
                             nlambda = nlambda,
                             intercept = constant,
                             parallel = F)


#Get weights for a second ENET
w.=(abs(coef(lasso1.s.1,s = Best_Lambda)[-1])+1/dim(X)[2])^(-1)


### second step, ENET with penalties (the "Adaptative" part)
rm(lasso1.s)
lasso1.s = foreach (t.x = 1:n.i, .combine=cbind , .packages =
                      c("data.table","magrittr")
) %bla% {
  if (is.null(seed)==F) seed.x = seed+t.x
  if (is.null(seed)==T) seed.x = NULL

  set.seed(seed.x)
  lasso1.s.x=glmnet::cv.glmnet(x = X, y = y,
                               family = family,
                               type.measure = "mse",
                               nfold = n.f,
                               alpha = alpha,
                               keep = TRUE,
                               nlambda = nlambda,
                               intercept = constant,
                               parallel = F,
                               penalty.factor = w.)

  cvMs.x = data.table(cvm = lasso1.s.x$cvm,
```

```
                        lambda = lasso1.s.x$lambda)  %>%
      setnames(c("cvm","lambda"),
               paste0(c("cvm","lambda"),t.x))

    gc()
    return(cvMs.x)
  }
  n.lasso1.s = names(lasso1.s)
  n.lasso1.s.c = n.lasso1.s[grepl("cvm", n.lasso1.s)]
  cvms.sA = data.table(lambda = lasso1.s[, lambda1],
                       cvm.m = lasso1.s[, mget(n.lasso1.s.c)] %>% rowMeans()
  )
  Best_ALambda = cvms.sA[order(cvm.m),lambda][1]
  if (is.null(seed)==F) seed.x = seed+1
  if (is.null(seed)==T) seed.x = NULL

  set.seed(seed.x)

  Alasso1.s.1=glmnet::cv.glmnet(x = X, y = y,
                                family = family,
                                type.measure = "mse",
                                nfold = n.f,
                                alpha = alpha,
                                keep = TRUE,
                                nlambda = nlambda,
                                intercept = constant,
                                parallel = F,
                                penalty.factor = w.)

### Finaly, save best aenet vars and coefficients escenario
alasso.glmnet.coef <- coef(Alasso1.s.1, s = Best_ALambda)

alasso.glmnet.coef = data.table("var_name" = row.names(alasso.glmnet.coef),
                                "coef" = as.numeric(alasso.glmnet.coef)    )

all_aenet = list("coefs" = alasso.glmnet.coef, "Best_ALambda" = Best_ALambda,
                 "cmvs" = cvms.s,
                 "Best_Lambda" = Best_Lambda,
                 "cmvsA" = cvms.sA)
return(all_aenet)
}
```

[ ]:
```
# 1) Cargar bases -----------------------
setwd(dir.in)
#Base de datos donde se encuentra la variable y de interés para SAE-NER y su conjunto de Xs
dat = readRDS("dat.rds")
setDT(dat)
#Objeto que contenga todas las potenciales variables Xs para el modelo
all.potential.Xs = readRDS("all.potential.Xs.rds")
```

2 Ada-ENET para selección de variables, ejemplo:

[ ]:
```
.Xs = dat[,mget(all.Xs.potential.v0)] %>% data.matrix()
.y = dat[,get("y")]  %>% data.matrix()

total_cores = parallel::detectCores()
invisible(gc())
cl = makeCluster(total_cores)
registerDoSNOW(cl)

Xs_SaeNer_aENET_fromAllPotXs_alpha05 =
aENET_idealXs (X = .Xs,y = .y,n.f = 10, constant = T, parallel = T, alpha = 0.5,
               nlambda = 100, family  = "gaussian", seed = 1, n.i = 100)
invisible(gc())
```

```
stopCluster(cl)
```

3 Bucle para diferentes valores de α

[ ]:
```
invisible(gc())
total_cores = parallel::detectCores()
cl = makeCluster(total_cores)
registerDoSNOW(cl)

Xs_aENET_all = list()
alphas = seq(0,1,0.1)

for (.a in alphas){
    name = paste("alpha = ".a)
    Xs_aENET_all[[name]] =
    aENET_idealXs(X = .Xs,y = .y,n.f = 10, constant = T, parallel = T, alpha = .a,
                  nlambda = 100, family  = "gaussian", seed = 1, n.i = 100)
    print(.diff.x)
    invisible(gc())
}
stopCluster(cl)
invisible(gc())
```