Métodos Estadísticos avanzados: Trabajo I

Santiago Aristizabal Toro, Sevastian Moreno Zapata, Danny Cardona Pineda

11 de octubre de 2019

1. Enunciado del problema

La conformación de 6 equipos de trabajo los cuales deben competir por obtener los mejores resultados términos de capacidad predictiva especifica (15 %), capacidad predictiva general (20 %) y selección de regresores (25 %). Para esto a los equipos de trabajo se les entregará 3 bases de datos, las cuales corresponden a variables dependientes con soportes continuo, binario y conteo. Sobre cada una de las bases de datos, se medirá el desempeño de cada uno de los equipos de trabajo en cada uno de los ítems de evaluación, tal que, el equipo con el mejor desempeño obtendrá una nota de 5, el segundo mejor 4.5, y así sucesivamente hasta 2.5. Cabe resaltar que, en caso de empate en alguna medida de desempeño, se promedian las notas respectivas a la posición superior e inferior, y se asigna esta nota a los respectivos equipos, tal que el siguiente equipo en desempeño se le asigna la nota inferior menos 0.5 unidades. El entregable debe ser un documento que establezca el proceso seguido para realizar la tarea encomendada para cada base de datos. Este archivo ser a utilizado para juzgar si el grupo de trabajo en cuestión merece una mejora en la nota obtenida en la competencia. Los otros archivos son los códigos de programación, tal que sean replicables sin ningún inconveniente, y que lean dos archivos de trabajo, el primero, la base de datos utilizada para escoger la metodología para realizar los pronósticos y la selección de variables, y el segundo, un archivo que contiene la base de datos sobre la cual se realiza la evaluación de la capacidad predictiva general, y especifica.

La capacidad predictiva general será el error cuadrático medio para las variables continuas y conteo, esta última debe tener un criterio para ser discretizada, y el accuracy (verdaderos positivos + verdaderos negativos dividido el tamaño muestral) para la variable binaria. La capacidad predictiva especifica ,sería la correcta clasificación de los valores inferiores y superiores a -1 para la variable continua. Para la variable binaria, será el área bajo la curva ROC. En el caso de la variable de conteo, será la correcta clasificación de los valores iguales a 0, y mayores a 0. En el caso de la selección de las variables cada correcta detección daría un valor de 1, y cada incorrecta clasificación restará 1.

2. Solución

Para la solución del problema se siguieron los siguientes pasos:

1. Lectura de datos

2. Análisis de correlaciones: Mediante la correlación de Spearman se trata de observar que tanta correlación hay entre las regresoras (indicios de multicolinealidad)

3. Análisis bivaridos entre cada regresor y la variable respuesta: Se usan pruebas chi-cuadrado para el caso de variables cualitativas y Mann-Whitney para el caso continuo. Este resultado se compara con lo obtenido con los métodos de selección de variables paramétricos y bayesianos.

4. Partición de datos en Train y Test (70 % y 30 % respectivamente)

5. Estandarización de regresores de naturaleza continua

6. Selección de variables con métodos frecuentistas: Ridge, Lasso, LARS y Elastic Net

7. Selección de variables con métodos bayesiano: Spike and Slab, Non-local Priors y shrinkage spike-slab.

Para encontrar los parámetros de regularización en los modelos frecuentistas, se hizo validación cruzada para encontrar los valores óptimos.

En todos los casos se entreno los modelos sobre el train y se probo y midió el MSE y capacidad predictiva específica (CPE) sobre los datos de test. Para la selección entre modelos se tuvieron en cuenta los siguientes aspectos:

(a) valores bajos del MSE

(b) CPE alto

(c) Modelo parsimonioso manteniendo (a) y (b)

(d) Estabilidad del modelo (mismo resultado al correr muchas veces)

2.1. Variable dependiente con soporte continuo

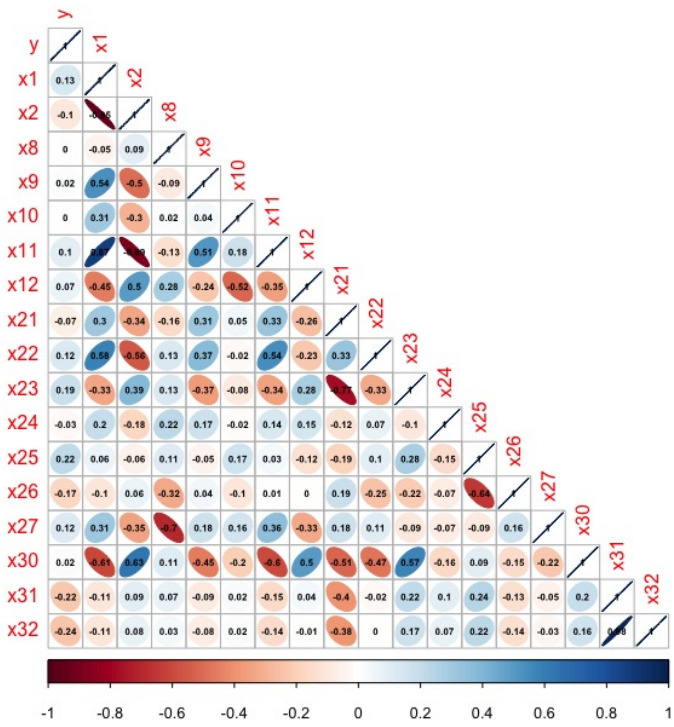

Figura 1: Correlación de Spearman entre regresoras de naturaleza continua

De la figura 1 se observa que existen correlaciones altas entre algunos regresores, por ejemplo entre X_1, X_2, X_{11}, X_{22} y $X30$. Este hecho es importante a la hora de la selección de variables, ya que se sabe que la multicolinealidad afecta seriamente el ajuste de modelos y algunos métodos de selección de variables son sensibles ante este fenómeno. Un caso particular son los métodos tradicionales como el backward y forward. Por esta razón se implementan formas alternativas a las tradicionales para la selección de variables, mediante metodologías que permiten lidiar con el problema de multicolinealidad, aunque este es un tema abierto y sigue siendo un problema a la hora de tratar de ajustar modelos adecuados a los datos.

De la tab la 1 se observa que regresores están más relacionado con la variable respuesta, aunque al ajustar de manera conjunta un modelo con todos, las variables estas relaciones puedan cambiar, se considera importante hacer los análisis bivariados.

Cuadro 1: Pruebas bivaridas de cada regresor y la variable respuesta ($\alpha = 0{,}2$).

Variable	Estadístico	Valor p
x7	415.00	0.14
x13	1252.00	0.12
x14	1021.00	0.19
x17	1383.00	0.17
x29	151.00	0.04
x1	11236.00	0.00
x2	11229.00	0.00
x8	11236.00	0.00
x9	9741.00	0.00
x10	11066.00	0.00
x11	11197.00	0.00
x12	8483.00	0.00
x22	4346.00	0.00
x23	4906.00	0.11
x24	4439.00	0.01
x25	11144.00	0.00
x26	11049.00	0.00
x27	11236.00	0.00
x30	9759.00	0.00
x31	5044.00	0.20

2.1.1. Modelos Frecuentistas

Se ajustaron varios modelos frecuentistas que permitieran una adecuada selección de variables considerando que existe alto nivel de colinealidad entre los regresores. Dentro de los modelos que se ajustaron está la regresión Ridge, Lasso, LARS y Elastic Net. Finalmente el modelo que se consideró el mejor entre los frecuentista fue el ajustado con LARS con el cual se seleccionaron las variables X_{23}, X_{25} y se observaron valores del $MSE = 4{,}980$ y $CPE = 73{,}333$. Sin embargo se observa que este modelo no es capaz de predecir valores negativos, por lo que se decidió finalmente no trabajar con ese modelo. Al final se adjunta el código de este procedimiento cómo anexo.

2.1.2. Modelos Bayesianos

Dentro de los métodos de selección bayesianos, el shrinkage spike-slab fue con el que mejores resultados se obtuvo en términos del MSE, CPE, parsimonia y estabilidad. Este método en particular fue el de mejor desempeño, ya que fue bastante estable comparado con el spike an slab convencional, a demás fue más parsimonioso que lo obtenido con non-local priors. A continuación se describen los resultados obtenidos:

- Variables seleccionadas: X_{23} y X_{25}
- MSE_{test} : 4,095
- CPE_{test} : 77,273 %

A continuación se presenta el código usado:

```r
library(caret)
library(dplyr)
library(readr)
library(basad)

df<- read_csv("~/datacontinuousstudents.csv")

corrplot::corrplot(cor(df[,-c(1:2)],method = "spearman"),method="ellipse",type="lower",
                   number.cex = .5,addCoef.col = "black")


#######=================================================================
## ==== data prepartion
set.seed(9877)
trainIndex <- createDataPartition(df$y, p = .7,
                                  list = FALSE,
                                  times = 1)
train <- df[ trainIndex,]
test <- df[-trainIndex,]

data_preparation<-function(df){
  cont_ind<-c(1,2,8:12,21:27,30:32)
  cont_var<-paste("x",cont_ind,sep = "")
  cate_ind<-c(3:7,13:20,28,29)
  cate_var<-paste("x",cate_ind,sep = "")

  df_cont<-df[,cont_var]
  df_cat<-df[,cate_var]

  df_cont_scale<-scale(df_cont)%>%as_tibble()
  X<-cbind(y=df$y,df_cat,df_cont_scale)
  res=list(X=X,df_cont=df_cont,df_cat=df_cat)
  return(res)
}


#=== ========
restrain<-data_preparation(train)

X_train<-restrain$X
train_cont<-restrain$df_cont
train_cat<-restrain$df_cat

x = model.matrix (y~.,X_train )
y = X_train[,1]

#=== ========
restest<-data_preparation(test)

X_test<-restest$X
test_cont<-restest$df_cont
test_cat<-restest$df_cat
```

```
# ============================
#************** Bivariate analysis ************************
# ============================
res_catego<-data.frame()
for (i in names(train_cat)) {
  pr<-wilcox.test(X_train[,"y"]~train_cat[,i][[1]], paired=FALSE)
  res_cat<-data_frame(x_cont=i,Wval=pr$statistic,pval=pr$p.value)
  res_catego<-rbind(res_catego,res_cat)
}
x_cat_selected<-res_catego%>%filter(pval<.2)


res_continu<-data.frame()
for (j in names(train_cont)) {
  pr<-wilcox.test(train_cont[,j][[1]],X_train[,"y"], paired=FALSE)
  res_cont<-data_frame(x_cont=j,Wval=pr$statistic,pval=pr$p.value)
  res_continu<-rbind(res_continu,res_cont)
}
x_cont_selected<-res_continu%>%filter(pval<.2)


###===== basad package: shrinkage spike-slab
iter<-50000
basad_model <- basad( x = X_train[,-1], y = y, prior.dist = "t", select.cri = "BIC",niter =
    iter) # for
print( basad_model )
coeff<-basad_model$est.B[basad_model$model.index]
estimate<-X_test$x23*coeff["x23"]+X_test$x25*coeff["x25"] +coeff["intercept"]
check<-tibble(pred=estimate,y=X_test$y)%>%mutate(err2=(pred-y)^2,check=case_when(
  ((y< -1)&(pred< -1))| ((y> -1)&(pred> -1))~1))
(indicador<-sum(check$check,na.rm = T)/nrow(X_test)*100)
(mse<-sum(check$err2)/nrow(X_test))
```

2.2. Variable dependiente con soporte binario

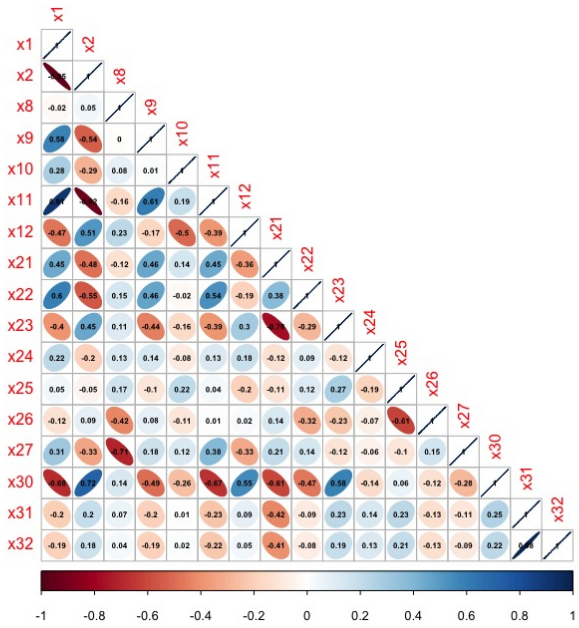

Figura 2: Correlación de Spearman entre regresoras de naturaleza continua

De la figura 2 se observa que existen correlaciones altas entre algunos regresores, por ejemplo entre X_1, X_2, X_{11}, X_{30}.

Cuadro 2: Pruebas bivaridas de cada regresor y la variable respuesta ($\alpha = 0{,}2$).

Variable	Estadístico	Valor p
x4	9.12	0.00
x15	1.83	0.18
x17	19.52	0.00
x18	2.08	0.15
x19	2.53	0.11
x1	1420.00	0.00
x2	3529.00	0.00
x9	2183.50	0.20
x10	1774.00	0.00
x11	1584.00	0.00
x12	3080.00	0.02
x22	1778.00	0.00
x26	2808.00	0.16
x27	2065.00	0.08
x30	3063.50	0.02

2.2.1. Modelos Frecuentistas

Para el caso de los datos binarios, la selección de modelos frecuentistas no tuvieron un buen desempeño, por ejemplo con regresión Lasso se obtuvo una precisión del 66.7 % con 19 variables, con elastic net se obtuvieron resultados similares.

2.2.2. Modelos Bayesianos

Se hizo una selección mediante spike and labs corriendo este muchas veces, y en el 100 % de la veces que se corrió, las variables $X_{24}, X_1, X_2, X_{19}, X_{11}, X_5$ siempre fueron seleccionadas, sin embargo los valores de sus coeficientes mostraron variaciones en cada corrida. Por esta razón se decidió ajustar un modelo lineal generalizado y un random forest con estas variables, para los cuales los resultados observados no fueron satisfactorios (precisiones menores al 65 %).

Finalmente se decidió ajustar un modelo Bayesiano generalizado con las variables X_1, X_{17}, X_4. La selección de estas tres variables obedece a un filtro realizado considerando solo aquellas variables que fueron preseleccionadas mediante Spike and Labs y/o aquellas que de manera individual tenían valores p menores a 0.01 (tabla 2). De estas se seleccionaron las que mejor resultados arrojaron de manera conjunta en términos del MSE, CPE, parsimonia y estabilidad. A continuación se presentan los resultados finales obtenidos.

- Variables seleccionadas: X_1, X_{17}, X_4
- $Precisión_{test}$: 75,556 %
- AUC_{test} : 60,560 %

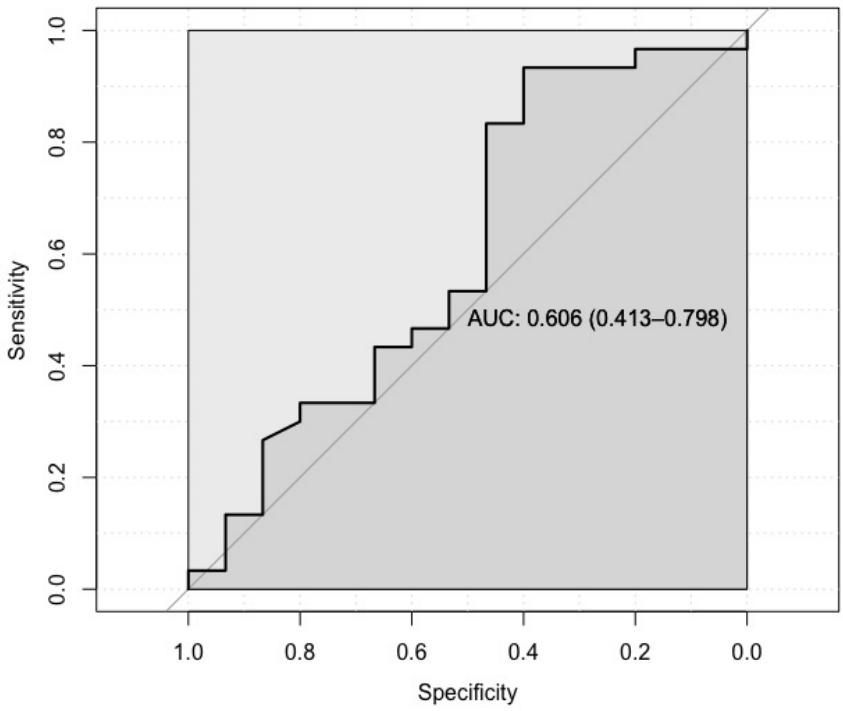

Figura 3: Capacidad predictiva del modelo (área bajo la curva ROC:AUC)

A continuación se presenta el código usado:

```
#### ============================================================================
## ================ Binary data
#### ============================================================================
library(readr)
library(dplyr)
library(MCMCpack)
library(brms) # This library use Stan
library(caret)
# ==========================
#************** data preparation ***********************
# ==========================

df <- read_csv("~/databinarystudents.csv")
df<-df[1:150,]

set.seed(9876)
trainIndex <- createDataPartition(df$yL, p = .7,
                                  list = FALSE,
                                  times = 1)
train <- df[ trainIndex,]
test <- df[-trainIndex,]

data_preparation<-function(df){
```

```
  cont_ind<-c(1,2,8:12,21:27,30:32)
  cont_var<-paste(rep("x",length(cont_ind)),cont_ind,sep = "")
  cate_ind<-c(3:7,13:20,28,29)
  cate_var<-paste(rep("x",length(cate_ind)),cate_ind,sep = "")

  df_cont<-df[,cont_var]
  df_cat<-df[,cate_var]

  df_cont_scale<-scale(df_cont)%>%as_tibble()
  X<-cbind(y=df$yL,df_cat,df_cont_scale)
  res=list(X=X,df_cont=df_cont,df_cat=df_cat)
  return(res)
}

#=== ========
restrain<-data_preparation(train)

X_train<-restrain$X
train_cont<-restrain$df_cont
train_cat<-restrain$df_cat

cont_ind<-c(1,2,8:12,21:27,30:32)
cont_var<-paste(rep("x",length(cont_ind)),cont_ind,sep = "")
cate_ind<-c(3:7,13:20,28,29)
cate_var<-paste(rep("x",length(cate_ind)),cate_ind,sep = "")

corrplot::corrplot(cor(X_train[,cont_var],method = "spearman"),method="ellipse",type="lower",
                   number.cex = .5,addCoef.col = "black")

#=== ========
restest<-data_preparation(test)

X_test<-restest$X
test_cont<-restest$df_cont
test_cat<-restest$df_cat

# ==========================
#************** Bivariate analysis ***********************
# ==========================

res_catego<-data.frame()
for (i in names(train_cat)) {
  tb<-table(df[,"yL"][[1]],df[,i][[1]])
  pr<-chisq.test(tb)
  res_cat<-data_frame(x_cat=i,chival=pr$statistic,pval=pr$p.value)
  res_catego<-rbind(res_catego,res_cat)
}
x_cat_selected<-res_catego%>%filter(pval<.2)

res_continu<-data.frame()
for (j in names(train_cont)) {
  pr<-wilcox.test(df[,j][[1]]~ df[,"yL"][[1]], paired=FALSE)
  res_cont<-data_frame(x_cont=j,Wval=pr$statistic,pval=pr$p.value)
```

```
  res_continu<-rbind(res_continu,res_cont)
}
x_cont_selected<-res_continu%>%filter(pval<.2)

# ============================
#************** Bayesian model ************************
# ============================

#===== Spike and labs

niter = 100000
m_select0 <- logit.spike(y ~ .,
                                    data = X_train,
                                    niter = niter)

fit<-m_select0
PIP <- colMeans(fit$beta != 0)
PIP
plot(fit,inclusion.threshold = .05)
plot(fit, "fit")
plot(fit, "residuals")
plot(fit, "size")
summary(fit)

y_pred<-ifelse(fit$fitted.probabilities<.5,0,1)
y_real<-X_train1[,'y']
table(y_pred,y_real)


# ================================
#************** Fit *******
# ================================

fit <- brm(y ~ x1+x17+x4, data = X_train,
           family = bernoulli(link = "logit"),iter = 2000)


summary(fit)
plot(fit)
pp = pp_check(fit)
pp + theme_bw()


##=====Predictions

newdata <- predict(fit, newdata = X_test,
                   type = "response")%>%as_tibble()
threshold<-0.5
newdata$class<-ifelse(newdata[,1]<=threshold,0,1)
(accuracy<-round(sum(X_test$y==newdata$class)/nrow(X_test)*100,3))

library(pROC)
ROC <- roc(test$yL,newdata$Estimate,
          smoothed = TRUE,
```

```
          ci=TRUE, ci.alpha=0.9, stratified=FALSE,
          plot=TRUE, auc.polygon=TRUE, max.auc.polygon=TRUE, grid=TRUE,
          print.auc=TRUE, show.thres=TRUE)
sens.ci <- ci.se(ROC)
plot(sens.ci, type="shape", col="lightblue")
plot(sens.ci, type="bars")
```

2.3. Variable dependiente con soporte conteo

2.3.1. Modelos Frecuentistas

Para el caso de los datos de conteo, se eligió ajustar finalmente un modelo Elastic Net. Esta decisión se tomo luego de haber seleccionado las variables por métodos frecuentistas y bayesianos, finalmente se encontró que bajo ambos enfoques las variables con mayores probabilidades de ser seleccionados fueron X_{25} y X_3. Es importante resaltar que preferimos ajustar este modelo ya con las variables seleccionadas, con el propósito de aplicar las penalizaciones L_1 y L_2.

Un aspecto importante dentro del análisis realizado es que solo hay de a un registro donde la variable respuesta es igual a 5,6 o 7. Por este motivo se decidió incluir estos registros dentro de los datos de entrenamiento sin importar si era o no seleccionados de manera aleatorio para tal fin, es decir que siempre se incluyeron dentro de los datos de entrenamiento. Esto se hizo con el fin de que el modelo pudiera capturar más variabilidad y de esta manera obtener un MSE menor al predecir sobre los datos de Test. Se probó haciendo este proceso de manera contraria, donde se observó que los valores del MSE fueron mayores. La otra opción era dejar que el azar dejara estos registros (conteos igual a 5,6,7) en Train o Test, pero esta opción afectó seriamente la estabilidad del modelo. También se exploró mediante asignación de pesos a las observaciones según la frecuencia relativa de aparición, para tratar de equilibrar la falta de balance, esta opción tampoco fue útil para el propósito del ejercicio.

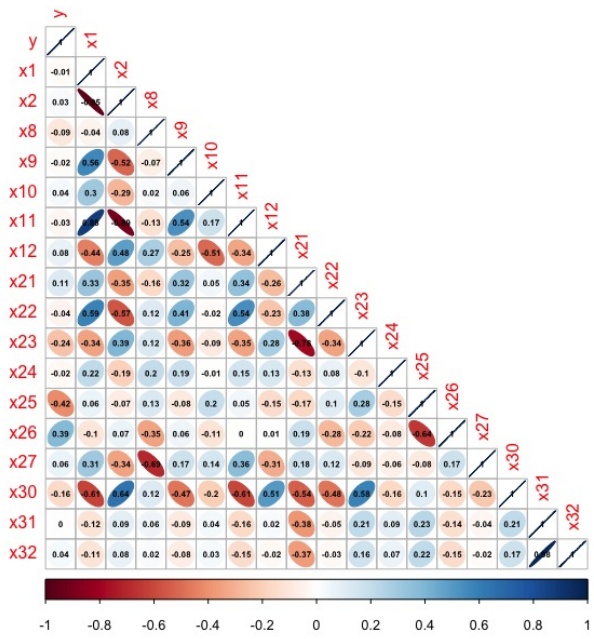

Figura 4: Correlación de Spearman entre regresoras de naturaleza continua

A continuación se presentan los resultados finales obtenidos.

- Variables seleccionadas: X_{25} y X_3
- MSE_{test} : 0,965
- CPE_{test} : 71,429 %

2.3.2. Modelos Bayesianos

Para obtener las variables a incluir en el modelo se hizo por medio de estimaciones shrinkage Spike-Slab, para el cual se observó que la "posterior probability"fue mayor para X_{25} y X_3.

A continuación se presenta el código usado:

```
#### ============================================================================
## ================= Modelo conteo
#### ============================================================================

library(caret)
library(dplyr)
library(readr)

df <- read_csv("~/datacountstudents.csv")
```

```
#######=========================================================================
## ==== data prepartion

df<-df %>% rename(y=yC)
df_small<-df %>% filter(y>4)
df<-df %>% filter(y<=4)

set.seed(9883)

trainIndex <- createDataPartition(df$y, p = .7,
                                  list = FALSE,
                                  times = 1)
train <- df[ trainIndex,]
test <- df[-trainIndex,]
train<-rbind(train,df_small)

data_preparation<-function(df){
  cont_ind<-c(1,2,8:12,21:27,30:32)
  cont_var<-paste("x",cont_ind,sep = "")
  cate_ind<-c(3:7,13:20,28,29)
  cate_var<-paste("x",cate_ind,sep = "")

  df_cont<-df[,cont_var]
  df_cat<-df[,cate_var]

  df_cont_scale<-scale(df_cont) %>% as_tibble()
  X<-cbind(y=df$y,df_cat,df_cont_scale)
  res=list(X=X,df_cont=df_cont,df_cat=df_cat)
  return(res)
}




#=== ========
restrain<-data_preparation(train)

X_train<-restrain$X
train_cont<-restrain$df_cont
train_cat<-restrain$df_cat


#=== ========
restest<-data_preparation(test)

X_test<-restest$X
test_cont<-restest$df_cont
test_cat<-restest$df_cat



# ============================
#************** Bivariate analysis ************************
# ============================
```

```
res_catego<-data.frame()
for (i in names(train_cat)) {
  pr<-wilcox.test(X_train[,"y"]~train_cat[,i][[1]], paired=FALSE)
  res_cat<-data_frame(x_cont=i,Wval=pr$statistic,pval=pr$p.value)
  res_catego<-rbind(res_catego,res_cat)
}
x_cat_selected<-res_catego%>%filter(pval<.2)


res_continu<-data.frame()
for (j in names(train_cont)) {
  pr<-wilcox.test(train_cont[,j][[1]],X_train[,"y"], paired=FALSE)
  res_cont<-data_frame(x_cont=j,Wval=pr$statistic,pval=pr$p.value)
  res_continu<-rbind(res_continu,res_cont)
}
x_cont_selected<-res_continu%>%filter(pval<.2)




#====== Set training control
y_val<-as.numeric(names(table(X_train$y)))
w<-1/table(X_train$y)

myweights<-w/sum(w)

model_weights<-NA
for (i in y_val) {
  model_weights<-ifelse(X_train$y==i,w[i+1],model_weights)
}



###===== basad package: shrinkage spike-slab

iter<-100000
basad_model <- basad( x = X_train[,-1], y = X_train$y, prior.dist = "t", select.cri =
    "BIC",niter = iter) # for
print( basad_model )



#============ Fit model

train_control <- trainControl(method = "repeatedcv",
                              number = 5,
                              repeats = 5,
                              search = "random",
                              verboseIter = F)

# Train the model

elastic <- train(y ~ .,
```

```
                data = X_train[,c("y","x25","x3")],
                method = "glmnet",
                tuneLength = 25,
                #weights = model_weights,
                trControl = train_control)


# Model coefficients
coef(elastic$finalModel, elastic$bestTune$lambda)
# Make predictions
predictions <- elastic %>% predict(X_test)
# Model prediction performance
#mse<-sum((predictions-X_test$y)^2)/nrow(X_test)
data.frame(
  RMSE = RMSE(predictions, X_test$y),
  MSE=RMSE(predictions, X_test$y)^2,
  Rsquare = R2(predictions, X_test$y)

)

check<-tibble(pred=predictions,y=X_test$y,pred_round=round(pred,0))%>%mutate(check=case_when(
  ((y==0)&(pred_round==0))| ((y>0 )&(pred_round> 0))~1
))

(precision<-sum(check$check,na.rm = T)/nrow(X_test)*100)
```

3. Conclusión

Los modelos presentados como finales, fueron el producto de analizar modelos Frecuentistas, Bayesianos y análisis bivariados.

Para los tres ejercicios se observó la existencia de multicolinealidad entre regresoras, por lo que la selección de los modelos por métodos convencionales (Forward, Backward, etc) fallan, y aún con las propuestas de modelos para abordar el problema que existen hoy, se concluye que el problema no está resuelto. Con todas las estrategias que se probaron para la selección de variables se encontraron dificultades decidiendo, más aún considerando que se debía mantener buenos valores para el MSE, CPE, parsimonia y estabilidad.

4. Anexos

4.1. Código adicional variable soporte continuo

```
#!/usr/bin/env python
# coding: utf-8

import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

```
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt

df=pd.read_csv('datacontinuousstudents.csv')
df.head()

pd.isnull(df).sum()

df_cat=pd.DataFrame(df[['x3','x4','x5','x6','x7','x13','x14','x15'
,'x16','x17','x18','x19','x20','x28','x29']])
df_cat.head()

df_num=df.drop(labels=['id','x3','x4','x5','x6','x7','x13','x14',
'x15','x16','x17','x18','x19','x20','x28','x29'],axis=1)
df_num.head()

df_num.describe()

corr=df_num.corr()

corr.style.background_gradient(cmap='coolwarm')


from sklearn.model_selection import KFold

train,test=train_test_split(df,test_size=0.3)
kftr=KFold(n_splits=10,random_state=None,shuffle=False)
kfts=KFold(n_splits=10,random_state=None,shuffle=False)
kftr.get_n_splits(train)
kfts.get_n_splits(test)
len(train),len(test)

train.head()

# ## Estandarizacion de VariablesTrain

vble_cat=train[['x3','x4','x5','x6','x7','x13','x14','x15','x16',
'x17','x18','x19','x20','x28','x29']]
len(vble_cat)

colm=vble_cat.columns.tolist()
vble_cat.head()

train_num=train.drop(labels=['id','y','x3','x4','x5','x6','x7',
'x13','x14','x15','x16','x17','x18','x19','x20','x28','x29'],axis=1)
train_num.head()

colmane=train_num.columns.values.tolist()
zscore_df = stats.zscore(train_num, axis=1)

train_esta=pd.DataFrame(zscore_df,columns=colmane)
len(train_esta)
```

```
train_esta=train_num.iloc[list(train_esta.index)]
train_esta.head()

train_esta=pd.concat([train['y'],train_esta],axis=1)
train_esta.head()

corr=train_esta.corr()

corr.style.background_gradient(cmap='coolwarm')

train_glo=pd.concat([train_esta,train[colm]],axis=1)
train_glo.head()

coln=train_glo.columns.values.tolist()
target=coln[0]
predictor=coln[1:len(coln)]
len(target),len(predictor)

# ## Estandarizacion de Variables Test

test.head()

test_num=test.drop(labels=['id','y','x3','x4','x5','x6','x7',
'x13','x14','x15','x16','x17','x18','x19','x20','x28','x29'],axis=1)
test_num.head()

colmanet=test_num.columns.values.tolist()
zscore_dft = stats.zscore(test_num, axis=1)

test_esta=pd.DataFrame(zscore_dft,columns=colmane)
len(test_esta)

test_esta=test_num.iloc[list(test_esta.index)]
test_esta.head()

test_esta=pd.concat([test['y'],test_esta],axis=1)
test_esta.head()

test_glo=pd.concat([test_esta,test[colm]],axis=1)
test_glo.head()

# ## ElasticNEt

from sklearn import metrics
from sklearn.model_selection import cross_val_score,cross_val_predict
from sklearn.metrics import accuracy_score, precision_score,
    classification_report,f1_score,recall_score
from sklearn.model_selection import KFold
import numpy as np
from sklearn.linear_model import ElasticNet

x=train_glo[predictor]
y=train_glo[target]
```

```
for i in np.arange(0,1,0.0001):
    lm_elas=ElasticNet(alpha=i, copy_X=True, fit_intercept=False, l1_ratio=0.75,
    max_iter=1000, normalize=False, positive=False, precompute=False,
    random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
    lm_elas.fit(x,y)
    predit=lm_elas.predict(test_glo[predictor])
    r=pd.Series(lm_elas.coef_,index=test_glo[predictor].columns)
    r=r[r!=0]
    #cv=KFold(n_splits=10,shuffle=True,random_state=1)
    #pre=np.round(cross_val_predict(lm_elas,x,y,cv=10))
    w=np.array([-1 if e<=-1 else 1 for e in predit])
    rs=np.array([-1 if e<=-1 else 1 for e in test_glo[target]])
    accuracy=accuracy_score(rs,w)
    print('accuracy',accuracy*100,'%')
    print('Las variables seleccionadas son:',len(r))
   # print('Rsquare elastic net i es=',i,'Rquere=',np.round(lm_elas.score(x,y)*100,2),'%')
    print('MSE=',metrics.mean_squared_error(test_glo[target],predit))
    print(' elastic net i es=',i)
    print('-----------------------------------------')


### Modelo Seleccionado Elasticnet

i=0.9999
lm_elas=ElasticNet(alpha=i, copy_X=True, fit_intercept=False, l1_ratio=0.75,
max_iter=1000, normalize=False, positive=False, precompute=False,
random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
lm_elas.fit(x,y)
predit=lm_elas.predict(test_glo[predictor])
r=pd.Series(lm_elas.coef_,index=test_glo[predictor].columns)
r=r[r!=0]
#cv=KFold(n_splits=10,shuffle=True,random_state=1)
#pre=np.round(cross_val_predict(lm_elas,x,y,cv=10))
w=np.array([-1 if e<=-1 else 1 for e in predit])
rs=np.array([-1 if e<=-1 else 1 for e in test_glo[target]])
accuracy=accuracy_score(rs,w)
print('accuracy',accuracy)
print('Las variables seleccionadas son:',len(r))
# print('Rsquare elastic net i es=',i,'Rquere=',np.round(lm_elas.score(x,y)*100,2),'%')
print('MSE=',metrics.mean_squared_error(test_glo[target],predit))
print(' elastic net i es=',i)
print(r)
print('-----------------------------------------')


### Metodo LassoLars

from sklearn import linear_model
from sklearn.linear_model import LassoLars
from sklearn.metrics import mean_squared_error
import numpy as np
for i in np.arange(0,0.0584,0.0001):
       reg=linear_model.LassoLars(alpha=i)
       LassoLars(alpha=i, copy_X=True, eps=..., fit_intercept=True,
```

```
         fit_path=True, max_iter=500, normalize=False, positive=False,
         precompute='auto', verbose=False)
        reg.fit(train_glo[predictor],train_glo[target])
        pc=reg.predict(test_glo[predictor])
        wc=np.array([-1 if e<=-1 else 1 for e in pc])
        rc=np.array([-1 if e<=-1 else 1 for e in test_glo[target]])

        #plc=np.array([0 if e==0 else 1 for e in pc])
        #rtc=np.array([0 if e==0 else 1 for e in test[target]])
        print('Datos para alpha =',i)
        #print(pd.crosstab(train[target],r,rownames=['actual'],colnames=['PREDICT']))
        exactitud=accuracy_score(wc,rc)
        print('La exactitud es del',exactitud*100,'%')
        #precision=precision_score(train[target],r,average='weighted')
        #print('La precision es del',precision*100,'%')
        #sensibilidad=recall_score(train[target],r,average='weighted')
        #print('La sensibilidad es del',sensibilidad*100,'%')
        #puntaje=f1_score(train[target],r,average='weighted')
        #print('El puntaje es del',puntaje*100,'%')
        ft=pd.Series(reg.coef_,index=test_glo[predictor].columns)
        a=ft[ft!=0]
        print('El numero de variables seleccionadas son:',len(a),'de',len(ft))
        mse=mean_squared_error(test_glo[target],pc)
        print('MSE',mse)
        print('-----------------------------------------------------------------')


### Modelo seleccionado para LassoLars

i=0.0492

reg=linear_model.LassoLars(alpha=i)
LassoLars(alpha=i, copy_X=True, eps=..., fit_intercept=True,
fit_path=True, max_iter=500, normalize=False, positive=False,
precompute='auto', verbose=False)
reg.fit(train_glo[predictor],train_glo[target])
pc=reg.predict(test_glo[predictor])
wc=np.array([-1 if e<=-1 else 1 for e in pc])
rc=np.array([-1 if e<=-1 else 1 for e in test_glo[target]])

#plc=np.array([0 if e==0 else 1 for e in pc])
#rtc=np.array([0 if e==0 else 1 for e in test[target]])
print('Datos para alpha =',i)
#print(pd.crosstab(train[target],r,rownames=['actual'],colnames=['PREDICT']))
exactitud=accuracy_score(wc,rc)
print('La exactitud es del',exactitud*100,'%')
#precision=precision_score(train[target],r,average='weighted')
#print('La precision es del',precision*100,'%')
#sensibilidad=recall_score(train[target],r,average='weighted')
#print('La sensibilidad es del',sensibilidad*100,'%')
#puntaje=f1_score(train[target],r,average='weighted')
#print('El puntaje es del',puntaje*100,'%')
ft=pd.Series(reg.coef_,index=test_glo[predictor].columns)
a=ft[ft!=0]
```

```
print('El numero de variables seleccionadas son:',len(a),'de',len(ft))
mse=mean_squared_error(test_glo[target],pc)
print('MSE',mse)
print(a)
print('-----------------------------------------------------------------')

i=0.0492
xx=train_glo[['x23','x25']]
xxx=test_glo[['x23','x25']]
reg=linear_model.LassoLars(alpha=i)
LassoLars(alpha=i, copy_X=True, eps=..., fit_intercept=True,
fit_path=True, max_iter=500, normalize=False, positive=False,
precompute='auto', verbose=False)
reg.fit(train_glo[predictor],train_glo[target])
pc=reg.predict(test_glo[predictor])
wc=np.array([-1 if e<=-1 else 1 for e in pc])
rc=np.array([-1 if e<=-1 else 1 for e in test_glo[target]])

#plc=np.array([0 if e==0 else 1 for e in pc])
#rtc=np.array([0 if e==0 else 1 for e in test[target]])
print('Datos para alpha =',i)
#print(pd.crosstab(train[target],r,rownames=['actual'],colnames=['PREDICT']))
exactitud=accuracy_score(wc,rc)
print('La exactitud es del',exactitud*100,'%')
#precision=precision_score(train[target],r,average='weighted')
#print('La precision es del',precision*100,'%')
#sensibilidad=recall_score(train[target],r,average='weighted')
#print('La sensibilidad es del',sensibilidad*100,'%')
#puntaje=f1_score(train[target],r,average='weighted')
#print('El puntaje es del',puntaje*100,'%')
ft=pd.Series(reg.coef_,index=test_glo[predictor].columns)
a=ft[ft!=0]
print('El numero de variables seleccionadas son:',len(a),'de',len(ft))
mse=mean_squared_error(test_glo[target],pc)
print('MSE',mse)
print(a)
print('-----------------------------------------------------------------')


# ## Metodo Lasso

from sklearn.linear_model import Lasso

for i in np.arange(0,1,0.0001):
    mlasso=linear_model.Lasso(alpha=i)
    Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=1000,
    normalize=False, positive=False, precompute=False, random_state=None,
    selection='cyclic', tol=0.0001, warm_start=False)
    mlasso.fit(train_glo[predictor],train_glo[target])
    pls=mlasso.predict(test_glo[predictor])

    wc=np.array([-1 if e<=-1 else 1 for e in pls])
    rc=np.array([-1 if e<=-1 else 1 for e in test_glo[target]])
    #pls=np.array([0 if e==0 else 1 for e in pls])
```

```
    #rtl=np.array([0 if e==0 else 1 for e in test[target]])
    #ft=pd.Series(mlasso.coef_,index=test[predictor].columns)
    print('Datos para alpha =',i)
    #print(pd.crosstab(train[target],r,rownames=['actual'],colnames=['PREDICT']))
    exactitud=accuracy_score(wc,rc)
    print('La exactitud es del',exactitud*100,'%')
    ftl=pd.Series(mlasso.coef_,index=test_glo[predictor].columns)
    al=ftl[ftl!=0]
    print('El numero de variables seleccionadas son:',len(al),'de',len(ftl))
    mse=mean_squared_error(test_glo[target],pls)
    print('MSE',mse)
   # print(al)
    print('-----------------------------------------------------------------')

i=0.9999
mlasso=linear_model.Lasso(alpha=i)
Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=1000,
normalize=False, positive=False, precompute=False, random_state=None,
selection='cyclic', tol=0.0001, warm_start=False)
mlasso.fit(train_glo[predictor],train_glo[target])
pls=mlasso.predict(test_glo[predictor])

wc=np.array([-1 if e<=-1 else 1 for e in pls])
rc=np.array([-1 if e<=-1 else 1 for e in test_glo[target]])
#pls=np.array([0 if e==0 else 1 for e in pls])
#rtl=np.array([0 if e==0 else 1 for e in test[target]])
#ft=pd.Series(mlasso.coef_,index=test[predictor].columns)
print('Datos para alpha =',i)
#print(pd.crosstab(train[target],r,rownames=['actual'],colnames=['PREDICT']))
exactitud=accuracy_score(wc,rc)
print('La exactitud es del',exactitud*100,'%')
ftl=pd.Series(mlasso.coef_,index=test_glo[predictor].columns)
al=ftl[ftl!=0]
print('El numero de variables seleccionadas son:',len(al),'de',len(ftl))
mse=mean_squared_error(test_glo[target],pc)
print('MSE',mse)
print(al)
print('-----------------------------------------------------------------')
```

4.2. Código adicional variable soporte de conteo

```
#!/usr/bin/env python
# coding: utf-8

# In[ ]:


import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

```
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt


df=pd.read_csv('\datacountstudents.csv')
df.head()

pd.isnull(df).sum()

df['yC'].value_counts()

sns.countplot(x='yC',data=df,palette='hls')
plt.savefig('count_plot')
plt.title('Descripcion de vble dependiente')
plt.show()

df_cat=pd.DataFrame(df[['yC','x3','x4','x5','x6','x7','x13','x14','x15','x16','x17',
    'x18','x19','x20','x28','x29']])
df_cat.head()

df_num=df.drop(labels=['id','yC','x3','x4','x5','x6','x7','x13','x14','x15','x16',
    'x17','x18','x19','x20','x28','x29'],axis=1)
df_num.head()

df_num.describe()

df_num=pd.concat([df['yC'],df_num],axis=1)

corr=df_num.corr()

corr.style.background_gradient(cmap='coolwarm')

from sklearn.model_selection import KFold


train,test=train_test_split(df,test_size=0.3)

kftr=KFold(n_splits=10,random_state=None,shuffle=False)
kfts=KFold(n_splits=10,random_state=None,shuffle=False)
kftr.get_n_splits(train)
kfts.get_n_splits(test)

len(train),len(test)

vble_cat=train[['yC','x3','x4','x5','x6','x7','x13','x14','x15','x16','x17','x18',
'x19','x20','x28','x29']]
len(vble_cat)

colm=vble_cat.columns.tolist()
vble_cat.head()

train_num=train.drop(labels=['id','yC','x3','x4','x5','x6','x7','x13','x14','x15','x16',
    'x17','x18','x19','x20','x28','x29'],axis=1)
```

```
train_num.head()

colmane=train_num.columns.values.tolist()
zscore_df = stats.zscore(train_num, axis=1)

train_esta=pd.DataFrame(zscore_df,columns=colmane)
len(train_esta)

train_esta=train_num.iloc[list(train_esta.index)]
train_esta.head()


# ## Estandarizacion de Variables Test

test_num=test.drop(labels=['id','yC','x3','x4','x5','x6','x7','x13','x14','x15',
    'x16','x17','x18','x19','x20','x28','x29'],axis=1)
test_num.head()

colmanet=test_num.columns.values.tolist()
zscore_dft = stats.zscore(test_num, axis=1)

test_esta=pd.DataFrame(zscore_dft,columns=colmane)
len(test_esta)

test_esta=test_num.iloc[list(test_esta.index)]
test_esta.head()

test_glo=pd.concat([test[colm],test_esta],axis=1)
test_glo.head()

#df_es=pd.concat(train_esta)

corr=train_esta.corr()

corr.style.background_gradient(cmap='coolwarm')

train_glo=pd.concat([train[colm],train_esta],axis=1)
train_glo.head()

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, precision_score,
    classification_report,f1_score,recall_score
from sklearn.model_selection import KFold
from sklearn.feature_selection import RFECV

coln=train_glo.columns.values.tolist()
target=coln[0]
predictor=coln[1:len(coln)]
len(target),len(predictor)


# ## Metodo ElastictNet
```

```
from sklearn import metrics
from sklearn.model_selection import cross_val_score,cross_val_predict
from sklearn.metrics import accuracy_score, precision_score,
    classification_report,f1_score,recall_score
from sklearn.model_selection import KFold
import numpy as np
from sklearn.linear_model import ElasticNet

x=train_glo[predictor]
y=train_glo[target]

for i in np.arange(0,1,0.0001):
    lm_elas=ElasticNet(alpha=i, copy_X=True, fit_intercept=False, l1_ratio=0.75,
    max_iter=1000, normalize=False, positive=False, precompute=False,
    random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
    lm_elas.fit(x,y)
    predit=np.round(lm_elas.predict(test_glo[predictor]))
    r=pd.Series(lm_elas.coef_,index=test_glo[predictor].columns)
    r=r[r!=0]
    w=np.array([0 if e==0 else 1 for e in predit])
    rs=np.array([0 if e==0 else 1 for e in test_glo[target]])
    accuracy=accuracy_score(rs,w)
    print('accuracy',accuracy)
    print('Las variables seleccionadas son:',len(r))
   # print('Rsquare elastic net i es=',i,'Rquere=',np.round(lm_elas.score(x,y)*100,2),'%')
    print('MSE=',metrics.mean_squared_error(test_glo[target],predit))
    print(' elastic net i es=',i)
    print('-----------------------------------------')


# ## Modelo seleccionado para ElasticNet

i=0.9999
lm_elas=ElasticNet(alpha=i, copy_X=True, fit_intercept=False, l1_ratio=0.75,
max_iter=1000, normalize=False, positive=False, precompute=False,
random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
lm_elas.fit(x,y)
predit=np.round(lm_elas.predict(test_glo[predictor]))
r=pd.Series(lm_elas.coef_,index=test_glo[predictor].columns)
r=r[r!=0]

#cv=KFold(n_splits=10,shuffle=True,random_state=1)
#pre=np.round(cross_val_predict(lm_elas,x,y,cv=10))
w=np.array([0 if e==0 else 1 for e in predit])
rs=np.array([0 if e==0 else 1 for e in test_glo[target]])
accuracy=accuracy_score(rs,w)
print('accuracy',accuracy)
print('Las variables seleccionadas son:',len(r))
# print('Rsquare elastic net i es=',i,'Rquere=',np.round(lm_elas.score(x,y)*100,2),'%')
print('MSE=',metrics.mean_squared_error(test_glo[target],predit))
print(' elastic net i es=',i)
print(r)
print('-----------------------------------------')
```

```
# ## Metodo LassoLars


from sklearn import linear_model
from sklearn.linear_model import LassoLars
from sklearn.metrics import mean_squared_error
import numpy as np
for i in np.arange(0,0.12380000000000001,0.0001):
       reg=linear_model.LassoLars(alpha=i)
       LassoLars(alpha=i, copy_X=True, eps=..., fit_intercept=True,
        fit_path=True, max_iter=500, normalize=False, positive=False,
        precompute='auto', verbose=False)
       reg.fit(train_glo[predictor],train_glo[target])
       p=reg.predict(test_glo[predictor])
       p=abs(np.round(p))
       pl=np.array([0 if e==0 else 1 for e in p])
       rt=np.array([0 if e==0 else 1 for e in test_glo[target]])
       print('Datos para alpha =',i)
       #print(pd.crosstab(train[target],r,rownames=['actual'],colnames=['PREDICT']))
       exactitud=accuracy_score(pl,rt)
       print('La exactitud es del',exactitud*100,'%')
       #precision=precision_score(train[target],r,average='weighted')
       #print('La precision es del',precision*100,'%')
       #sensibilidad=recall_score(train[target],r,average='weighted')
       #print('La sensibilidad es del',sensibilidad*100,'%')
       #puntaje=f1_score(train[target],r,average='weighted')
       #print('El puntaje es del',puntaje*100,'%')
       ft=pd.Series(reg.coef_,index=test_glo[predictor].columns)
       a=ft[ft!=0]
       print('El numero de variables seleccionadas son:',len(a),'de',len(ft))
       mse=mean_squared_error(test_glo[target],p)
       print('MSE',mse)
       print('-----------------------------------------------------------------')


# ## Modelo seleccionado para LassoLars


i=0.0352
xx=train_glo[['x3','x25']]
xxx=test_glo[['x3','x25']]
reg=linear_model.LassoLars(alpha=i)
LassoLars(alpha=i, copy_X=True, eps=..., fit_intercept=True,
fit_path=True, max_iter=500, normalize=False, positive=False,
precompute='auto', verbose=False)
reg.fit(xx,train_glo[target])
p=reg.predict(xxx)
p=abs(np.round(p))
pl=np.array([0 if e==0 else 1 for e in p])
rt=np.array([0 if e==0 else 1 for e in test_glo[target]])
print('Datos para alpha =',i)
exactitud=accuracy_score(pl,rt)
print('La exactitud es del',exactitud*100,'%')
```

```
ft=pd.Series(reg.coef_,index=xxx.columns)
a=ft[ft!=0]
print('El numero de variables seleccionadas son:',len(a),'de',len(ft))
mse=mean_squared_error(test_glo[target],p)
print('MSE',mse)
print(a)
print('--------------------------------------------------------------')


# ## Metodo Lasso

# In[ ]:


from sklearn.linear_model import Lasso

for i in np.arange(0,1,0.0001):
    mlasso=linear_model.Lasso(alpha=i)
    Lasso(alpha=i, copy_X=True, fit_intercept=True, max_iter=1000,
    normalize=False, positive=False, precompute=False, random_state=None,
    selection='cyclic', tol=0.0001, warm_start=False)
    mlasso.fit(train_glo[predictor],train_glo[target])
    pls=mlasso.predict(test_glo[predictor])
    pls=abs(np.round(pls))
    pls=np.array([0 if e==0 else 1 for e in pls])
    rtl=np.array([0 if e==0 else 1 for e in test_glo[target]])
    ft=pd.Series(mlasso.coef_,index=test_glo[predictor].columns)
    print('Datos para alpha =',i)
    #print(pd.crosstab(train[target],r,rownames=['actual'],colnames=['PREDICT']))
    exactitud=accuracy_score(pls,rtl)
    print('La exactitud es del',exactitud*100,'%')
    ftl=pd.Series(mlasso.coef_,index=test_glo[predictor].columns)
    al=ftl[ftl!=0]
    print('El numero de variables seleccionadas son:',len(al),'de',len(ftl))
    mse=mean_squared_error(test_glo[target],pls)
    print('MSE',mse)
   # print(al)
    print('--------------------------------------------------------------')


# ## Modelo Selecciondado para Lasso

# In[ ]:


i=0.9999
mlasso=linear_model.Lasso(alpha=i)
Lasso(alpha=i, copy_X=True, fit_intercept=True, max_iter=1000,
normalize=False, positive=False, precompute=False, random_state=None,
selection='cyclic', tol=0.0001, warm_start=False)
mlasso.fit(train_glo[predictor],train_glo[target])
pls=mlasso.predict(test_glo[predictor])
pls=abs(np.round(pls))
pls=np.array([0 if e==0 else 1 for e in pls])
```

```
rtl=np.array([0 if e==0 else 1 for e in test_glo[target]])
ft=pd.Series(mlasso.coef_,index=test_glo[predictor].columns)
print('Datos para alpha =',i)
#print(pd.crosstab(train[target],r,rownames=['actual'],colnames=['PREDICT']))
exactitud=accuracy_score(pls,rtl)
print('La exactitud es del',exactitud*100,'%')
ftl=pd.Series(mlasso.coef_,index=test_glo[predictor].columns)
al=ftl[ftl!=0]
print('El numero de variables seleccionadas son:',len(al),'de',len(ftl))
mse=mean_squared_error(test_glo[target],pls)
print('MSE',mse)
print(al)
print('-----------------------------------------------------------------')
```