Trabajo Final - Paolo Ciancaglini.ipynb

MDOULO 2 – FERNANDO OXA

import pandas as pd

import numpy as np

from sklearn.linear\_model import LinearRegression, Ridge, RidgeCV, Lasso, LassoCV, ElasticNet, ElasticNetCV

from sklearn.metrics import mean\_squared\_error

from sklearn.preprocessing import StandardScaler, PolynomialFeatures

from sklearn.model\_selection import cross\_val\_score, GridSearchCV, train\_test\_split, RandomizedSearchCV

from sklearn.feature\_selection import RFE, RFECV

import seaborn as sns

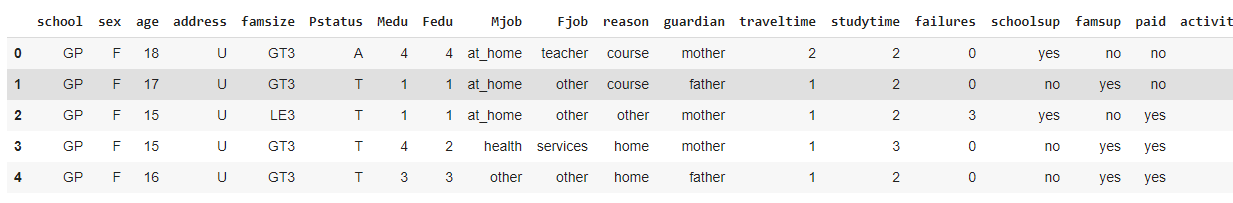
import matplotlib.pyplot as plt

import statsmodels.api as sm

#como el objetivo es solamente la prediccion, nos quedamos con esas dos variables dicotomicas (race/ethnicity), que son utiles para predecir valores de la variable objetivo.

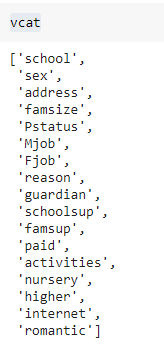
mathfinalgrade=pd.read\_csv('/content/drive/My Drive/Colab Notebooks/student-mat.csv')

mathfinalgrade.head()



vcat = list(mathfinalgrade[['school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob', 'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', 'romantic']])

vnum = set(list(mathfinalgrade)).difference(set(vcat)).difference({'G3'})



def modelos(X, Y, norm=False, cv=5):

  regs = cross\_val\_score(LinearRegression(normalize=norm), X, Y, scoring='neg\_mean\_squared\_error', cv = cv)

  ar = RidgeCV(normalize = norm).fit(X, Y).alpha\_

  ridges = cross\_val\_score(Ridge(alpha=ar, normalize=norm), X, Y, scoring='neg\_mean\_squared\_error', cv = cv)

  al = LassoCV(normalize = norm).fit(X, Y).alpha\_

  lasos = cross\_val\_score(Lasso(alpha=ar, normalize=norm), X, Y, scoring='neg\_mean\_squared\_error', cv = cv)

  ae = ElasticNetCV(normalize=norm).fit(X, Y).alpha\_

  ens = cross\_val\_score(ElasticNet(alpha=ae, normalize=norm), X, Y, scoring='neg\_mean\_squared\_error', cv = cv)

  print('reg lineal: ', np.mean(regs))

  print('hiperparam ridge: ', ar)

  print('reg ridge: ', np.mean(ridges))

  print('hiperparam lasso: ', al)

  print('reg lasso: ', np.mean(lasos))

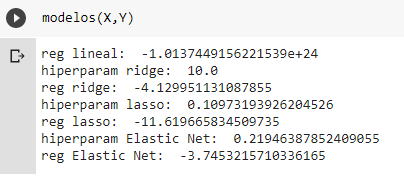
  print('hiperparam Elastic Net: ', ae)

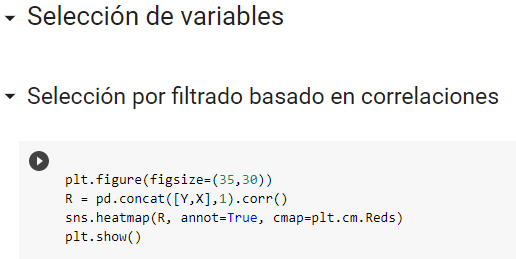
  print('reg Elastic Net: ', np.mean(ens))

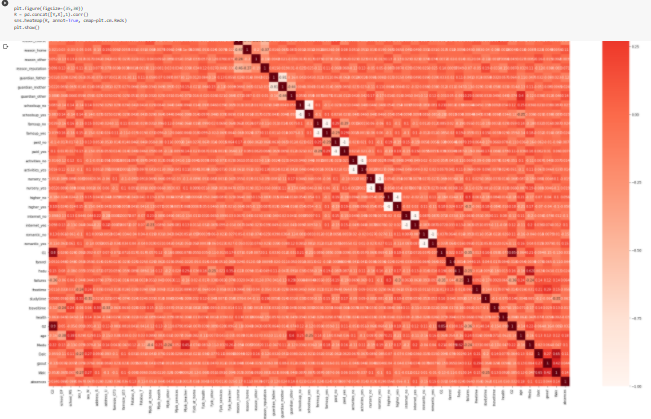
Dicos = pd.get\_dummies(mathfinalgrade[vcat],drop\_first=False)

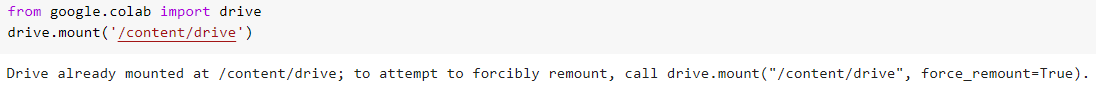
X = pd.concat([Dicos,mathfinalgrade[vnum]],axis=1)

Y = mathfinalgrade['G3']







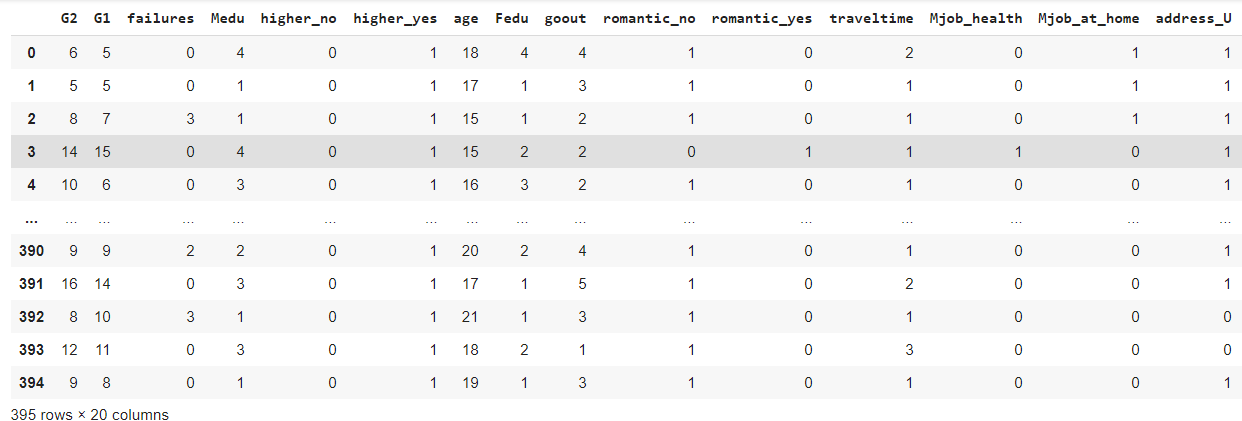


corrs = abs(R['G3']).sort\_values(ascending=False)

Xr = X[corrs[1:21].index.tolist()]

# se usan las variables con datos mayores a 0.10 en su correlacion con math score, que son 20.

Xr



corrs

G3 1.000000

G2 0.904868

G1 0.801468

failures 0.360415

Medu 0.217147

higher\_no 0.182465

higher\_yes 0.182465

age 0.161579

Fedu 0.152457

goout 0.132791

romantic\_no 0.129970

romantic\_yes 0.129970

traveltime 0.117142

Mjob\_health 0.116158

Mjob\_at\_home 0.115634

address\_U 0.105756

address\_R 0.105756

sex\_F 0.103456

sex\_M 0.103456

paid\_yes 0.101996

paid\_no 0.101996

reason\_course 0.098950

internet\_yes 0.098483

internet\_no 0.098483

studytime 0.097820

Mjob\_other 0.096477

reason\_reputation 0.095692

Fjob\_teacher 0.095374

guardian\_other 0.087774

schoolsup\_no 0.082788

schoolsup\_yes 0.082788

famsize\_LE3 0.081407

famsize\_GT3 0.081407

Mjob\_services 0.078429

health 0.061335

Pstatus\_A 0.058009

Pstatus\_T 0.058009

Mjob\_teacher 0.057712

Fjob\_health 0.057111

Dalc 0.054660

Fjob\_other 0.053483

reason\_other 0.052008

Walc 0.051939

nursery\_no 0.051568

nursery\_yes 0.051568

famrel 0.051363

school\_MS 0.045017

school\_GP 0.045017

famsup\_yes 0.039157

famsup\_no 0.039157

absences 0.034247

guardian\_father 0.032493

guardian\_mother 0.022338

reason\_home 0.021359

Fjob\_services 0.016108

activities\_no 0.016100

activities\_yes 0.016100

Fjob\_at\_home 0.013385

freetime 0.011307

Name: G3, dtype: float64

modelos(X,Y)

#podemos ver que en nuestro modelo normal, el mejor modelo es de ElasticNet, con valor de 3.74.

#Nuestra regresion lineal esta muy desalineada

reg lineal: -1.0137449156221539e+24

hiperparam ridge: 10.0

reg ridge: -4.129951131087855

hiperparam lasso: 0.10973193926204526

reg lasso: -11.619665834509735

hiperparam Elastic Net: 0.21946387852409055

reg Elastic Net: -3.7453215710336165

modelos(Xr,Y) #nuestra regresion lineal ahora nos muestra datos mas cercanos a sus pares.

#En el modelo Selección por filtrado basado en correlaciones,  la mejor es la MSE de ElasticNet. Pero la ElasticNet del modelo normal es mejor que la del modelo Selección por filtrado basado en correlaciones.

#Tambien se puede observar que despues de seleccionar variables, la regresion lineal se asemeja mucho a los MSE de las otras regresiones.

reg lineal: -4.043212555795147

hiperparam ridge: 10.0

reg ridge: -4.006973399705279

hiperparam lasso: 0.2204770338007646

reg lasso: -11.619665834509735

hiperparam Elastic Net: 0.4112352392669558

reg Elastic Net: -3.849411850390511

Selección stepwise backward

#metodo Stepside BACKWARD

cols = list(X.columns)

while (len(cols)>0):

  X\_1 = sm.add\_constant(X[cols])

  modelo = sm.OLS(Y, X\_1).fit()

  if(max(modelo.pvalues)>0.05):

    cols.remove(modelo.pvalues[1:].idxmax())

  else:

    break

print(cols)

['G1', 'famrel', 'G2', 'absences']

modelos(X[cols],Y) #Las MSE del metodo stepwise backward son un poco mas bajas que en los anteriores modelos. La regression Lasso se mantiene con un valor de 11.

#SE COMPARA el resultado con los resultados de los modelos anteriores. Hasta ahora, el ridge del metodo stepwise backward es el mejor (MSE=3.7314)

reg lineal: -3.734175238504021

hiperparam ridge: 10.0

reg ridge: -3.7314463982174844

hiperparam lasso: 0.015554218875180263

reg lasso: -11.619665834509735

hiperparam Elastic Net: 0.04409540676015293

reg Elastic Net: -3.732682344080112

## Recursive Feature Selection

rfecv = RFECV(LinearRegression(), scoring='neg\_mean\_squared\_error', cv=5).fit(X, Y)

rfecv.n\_features\_

rfecv

RFECV(cv=5,

estimator=LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None,

normalize=False),

min\_features\_to\_select=1, n\_jobs=None, scoring='neg\_mean\_squared\_error',

step=1, verbose=0)

rfecv.grid\_scores\_ #aqui podemos observar todos los MSE Negativos.

#a medida que se utilizan mas variables, aumenta el MSE.

array([-7.24480998e+00, -7.31035400e+00, -7.36377771e+00, -7.43429186e+00,

-7.37916651e+00, -7.37106709e+00, -7.43964545e+00, -7.44430839e+00,

-7.43410764e+00, -7.47598147e+00, -7.55963510e+00, -7.32780372e+00,

-7.41523065e+00, -7.67186466e+00, -7.55945667e+00, -7.52715551e+00,

-7.46937796e+00, -7.46921375e+00, -7.71971177e+00, -7.68222378e+00,

-7.83516768e+00, -7.68285645e+00, -7.71318699e+00, -7.67516558e+00,

-7.62851105e+00, -7.67428666e+00, -7.60084200e+00, -7.61397056e+00,

-7.66073969e+00, -7.57201199e+00, -7.62049749e+00, -7.69842321e+00,

-7.61588085e+00, -7.74230868e+00, -7.94082200e+00, -7.97408804e+00,

-8.08477955e+00, -8.01361004e+00, -7.99248204e+00, -8.08872640e+00,

-8.21047678e+00, -1.96472479e+19, -5.62377920e+26, -5.01429606e+24,

-6.19114483e+22, -1.95549661e+23, -4.26903020e+24, -2.85427222e+24,

-7.07003099e+23, -2.38252763e+23, -2.41164483e+22, -2.89433297e+23,

-1.78345207e+24, -6.11223118e+23, -5.26846141e+24, -6.58389942e+22,

-3.59289182e+23, -1.01374492e+24])

plt.figure()

plt.xlabel("Número de atributos seleccionados")

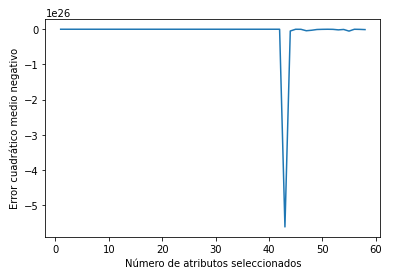
plt.ylabel("Error cuadrático medio negativo")

plt.plot(range(1, len(rfecv.grid\_scores\_) +1), rfecv.grid\_scores\_)

plt.show()

#aqui podemos observar que los incrementos son apenas perceptibles.

#Si, hay un salto significativo de la 6ta a la 8va variable. Si queremos reducir el numero de variables de atributos, podemos usar 8 variables.



rfe = RFE(LinearRegression(), 41) #con el numero de variables optimo (8), llamo a RFE.

X\_rfe = rfe.fit\_transform(X, Y) #LUEGO me va a devolver la matriz X con las variables seleccionadas con este metodo. Nos da un resultado en una matriz.

print(rfe.support\_) #me devuelve True si es la variable seleccionada, y False si no lo es.

print(rfe.ranking\_) #todos los True son 1. los False no son 1, dando asi un numero segun la importancia de las variables. La variable mas importante despues del 1 es 2, luego 3.

#en este caso, vamos a utilizar solamente las variables True (1, las mas importantes)

X\_rfe

[ True True False True False False False False True True False True

True True False False True False True True True True True True

True True False True True True True False False True True True

True True True False False True True True True True True False

True True True True True True True False True False]

[ 1 1 2 1 8 3 16 14 1 1 18 1 1 1 15 12 1 4 1 1 1 1 1 1

1 1 9 1 1 1 1 13 5 1 1 1 1 1 1 6 10 1 1 1 1 1 1 7

1 1 1 1 1 1 1 17 1 11]

array([[1, 0, 0, ..., 4, 1, 1],

[1, 0, 0, ..., 1, 1, 1],

[1, 0, 0, ..., 1, 2, 3],

...,

[0, 1, 1, ..., 1, 3, 3],

[0, 1, 1, ..., 3, 3, 4],

[0, 1, 1, ..., 1, 3, 3]])

modelos(X\_rfe,Y)

# En el modelo Recursive Feature Selection, la mejor MSE es la de reg ridge (MSE=3.97)

#Sin embargo, el ridge del metodo stepwise backward es el mejor (MSE=3.7314) hasta el momento.

reg lineal: -4.124680822906975

hiperparam ridge: 10.0

reg ridge: -3.971696742637147

hiperparam lasso: 0.09543924472847903

reg lasso: -11.619665834509735

hiperparam Elastic Net: 0.16601637582637838

reg Elastic Net: -3.8127394578436187

## Selección por método Lasso

regl = LassoCV().fit(X,Y)

print(regl.alpha\_, regl.score(X,Y))

cf = pd.Series(regl.coef\_, index = X.columns)

varsec = cf[cf!=0].index.tolist()

varsec #tenemos 7 variables seleccionadas por lasso

0.10973193926204526 0.8331225530194339

['G1', 'famrel', 'health', 'G2', 'age', 'Walc', 'absences']

regl.coef\_

array([-0. , 0. , -0. , 0. , 0. ,

-0. , -0. , 0. , 0. , -0. ,

-0. , -0. , 0. , 0. , 0. ,

-0. , 0. , 0. , -0. , 0. ,

0. , -0. , 0. , 0. , -0. ,

0. , -0. , -0. , 0. , -0. ,

0. , -0. , 0. , 0. , -0. ,

0. , -0. , -0. , 0. , 0. ,

-0. , 0. , -0. , 0.15094821, 0.21846052,

-0. , -0. , 0. , -0. , 0. ,

0.00898727, 0.97964939, -0.12753077, 0. , 0. ,

0. , 0.03973994, 0.03828369])

modelos(X[varsec],Y)  #usamos las 7 variables seleccionadas por varsec.

# En el modelo Lasso, la mejor MSE es la de reg ridge (MSE=3.6556)

#El ridge del metodo stepwise backward (MSE=3.7314), es mejorado por la reg Ridge del metodo Lasso.

reg lineal: -3.65678524433944

hiperparam ridge: 10.0

reg ridge: -3.6556645706748605

hiperparam lasso: 0.015554218875180263

reg lasso: -11.619665834509735

hiperparam Elastic Net: 0.03110843775036053

reg Elastic Net: -3.6582420948752876

## Lasso normalizado

regln = LassoCV(normalize=True).fit(X,Y)

print(regln.alpha\_, regln.score(X,Y))

cfn = pd.Series(regln.coef\_, index = X.columns)

varsecn = cfn[cfn!=0].index.tolist()

varsecn #podemos ver que aqui el lasso normalizado ha seleccionado 8 variables. Usando esta normalizacion, tenemos otro conjunto de variables.

0.008410060053398269 0.8280389983314642

['Fjob\_services',

'romantic\_no',

'G1',

'famrel',

'failures',

'G2',

'age',

'absences']

  modelos(X[varsecn],Y,norm=True)

  # En el modelo Lasso Normalizado, la mejor MSE es la de reg lineal (MSE=3.69)

  #Sin embargo, el ridge del metodo Lasso es el mejor (MSE=3.6556) hasta el momento.

reg lineal: -3.6976635234986546

hiperparam ridge: 0.1

reg ridge: -3.848877731026004

hiperparam lasso: 0.0020832360910743243

reg lasso: -7.133034249341273

hiperparam Elastic Net: 0.0004166472182148645

reg Elastic Net: -3.7737777913971464

## Elastic Net

elast = ElasticNetCV().fit(X,Y)

print(elast.alpha\_, elast.score(X,Y))

cfe = pd.Series(elast.coef\_, index = X.columns)

varsee = cfe[cfe!=0].index.tolist()

varsee #en este caso, tenemos 8 variables seleccionadas por el metodo ElasticNet.

0.21946387852409055 0.8325737834956021

['G1', 'famrel', 'failures', 'health', 'G2', 'age', 'Walc', 'absences']

modelos(X[varsee],Y)

# En el modelo ElasticNet, la mejor MSE es la de reg ElasticNet (MSE=3.6873)

#Sin embargo, el ridge del metodo Lasso es el mejor (MSE=3.6556) hasta el momento.

reg lineal: -3.705931501651901

hiperparam ridge: 10.0

reg ridge: -3.698894545923705

hiperparam lasso: 0.03593226298514801

reg lasso: -11.619665834509735

hiperparam Elastic Net: 0.06702109744207416

reg Elastic Net: -3.687297928495068

CONCLUSION: Se construyó un algoritmo para estimar varios modelos de regresión, con el objetivo de predecir la nota final (G3). Segun la comparación de los MSE de los diferentes modelos, la regresión ridge del metodo Lasso es el mejor (MSE=3.6556), siendo el MSE mas bajo de todos los modelos.

# **Evaluación del modelo (parte 2)**

## Variables del modelo

vars\_cat\_mod = ['school', 'sex', 'address', 'famsize', 'Pstatus', 'Mjob', 'Fjob', 'reason', 'guardian', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery', 'higher', 'internet', 'romantic']

vars\_cuant\_mod = ['Dalc', 'Fedu', 'G1', 'G2', 'Medu', 'Walc', 'absences', 'age', 'failures', 'famrel', 'freetime', 'goout', 'health', 'studytime','traveltime']

X = mathfinalgrade[vars\_cuant\_mod]

Y = mathfinalgrade.G3

## Conjuntos de entrenamiento y de prueba

Xen, Xpr, Yen, Ypr = train\_test\_split(X, Y)

np.shape(Xen)

np.shape(Xpr)

Xen, Xpr, Yen, Ypr = train\_test\_split(X, Y, test\_size=0.2)

## Evaluación dentro y fuera de la muestra de entrenamiento

reg = LinearRegression()

reg.fit(Xen, Yen)

Yen\_p = reg.predict(Xen)

Ypr\_p = reg.predict(Xpr)

print('MSE entrenamiento:', mean\_squared\_error(Yen\_p, Yen), '        MSE prueba', mean\_squared\_error(Ypr\_p, Ypr))

MSE entrenamiento: 3.2315063225285616 MSE prueba 4.285212118238835

# **Validación cruzada**

reg = LinearRegression()

mses = cross\_val\_score(reg, X, Y, scoring='neg\_mean\_squared\_error', cv=5)

mses = -mses

print(mses)

print(np.mean(mses))

[1.90444068 2.72594361 3.43076818 4.57525204 6.57870861]

3.843022624431223

# **Métodos de contracción**

## Regresión Ridge

lambdas = {'alpha': np.arange(1, 301)}

ridges = GridSearchCV(Ridge(), lambdas, scoring='neg\_mean\_squared\_error', cv=5)

ridges.fit(X, Y)

print(ridges.best\_params\_)

print(ridges.best\_score\_)

#el mejor parametro de lambda es el 99.

#Comparamos el dato resultado 3.7721 (mejor MSE) con el dato de validacion cruzada 3.8430 (MSE mean), y vemos que nuestro nuevo dato de estimacion es un poco mas pequenho. Tiene un error cuadratico medio menor.

{'alpha': 99}

-3.772109923853402

## Regresión Lasso

alfas = {'alpha': np.arange(1, 501)}

lasos = GridSearchCV(Lasso(), alfas, scoring='neg\_mean\_squared\_error', cv=5)

lasos.fit(X, Y)

print(lasos.best\_params\_)

print(lasos.best\_score\_)

#en el resultado podemos observar que mientras mas subimos el rango, nuestro valor mejor MSE baja cada vez sube mas.

#el mejor parametro es el 1, con MSE de 3.89

{'alpha': 1}

-3.8918950427094914

# **Comparando varios modelos**

## Función comparativa

def modelo2(X,Y):

  reg = LinearRegression()

  mses = cross\_val\_score(reg, X, Y, scoring='neg\_mean\_squared\_error', cv=5)

  lambdas = {'alpha': np.arange(1, 101)}

  rid = Ridge(normalize = True)

  ridges = GridSearchCV(rid, lambdas, scoring='neg\_mean\_squared\_error', cv=5)

  ridges.fit(X, Y)

  las = Lasso(normalize = True)

  lasos = GridSearchCV(las, lambdas, scoring='neg\_mean\_squared\_error', cv=5,)

  lasos.fit(X, Y)

  print('reg lineal: ', np.mean(mses))

  print('hiperparam ridge: ', ridges.best\_params\_)

  print('reg ridge: ',ridges.best\_score\_)

  print('hiperparam lasso: ', lasos.best\_params\_)

  print('reg lasso: ',lasos.best\_score\_)

modelo2(X,Y)

reg lineal: -3.843022624431223

hiperparam ridge: {'alpha': 1}

reg ridge: -6.366366684683913

hiperparam lasso: {'alpha': 1}

reg lasso: -21.256092773593977

## Añadiendo variables Dicotómicas

Dicos = pd.get\_dummies(mathfinalgrade[vars\_cat\_mod], drop\_first = False) #Transformamos las variables categoricas en dicotomicas.

X = pd.concat([Dicos, mathfinalgrade[vars\_cuant\_mod]], axis=1)

Y = mathfinalgrade.G3

modelo2(X, Y)

reg lineal: -8.249991581130064e+23

hiperparam ridge: {'alpha': 1}

reg ridge: -6.74419584302664

hiperparam lasso: {'alpha': 1}

reg lasso: -21.256092773593977

## Añadiendo variables polinomiales

Y = mathfinalgrade.G3

for i in range(1,4):

  Xi = PolynomialFeatures(i).fit\_transform(mathfinalgrade[vars\_cuant\_mod])

  print('Modelo polinomial para grado', i)

  modelo2(Xi, Y)

  #Podemos observar que el mejor MSE es del modelo regresion lineal, del modelo polinomial de grado 1 (MSE=3.8430). En los modelos de grado 2 y 3, la regresion ridge es la mejor de las 3 regresiones.

Modelo polinomial para grado 1

reg lineal: -3.8430226244312267

hiperparam ridge: {'alpha': 1}

reg ridge: -6.366366684683915

hiperparam lasso: {'alpha': 1}

reg lasso: -21.256092773593977

Modelo polinomial para grado 2

reg lineal: -9.310796066015063

hiperparam ridge: {'alpha': 1}

reg ridge: -4.135055444881502

hiperparam lasso: {'alpha': 1}

reg lasso: -21.256092773593977

Modelo polinomial para grado 3

reg lineal: -276.9313479266491

hiperparam ridge: {'alpha': 1}

reg ridge: -4.003136281445101

hiperparam lasso: {'alpha': 1}

reg lasso: -21.256092773593977

## Combinando dicotómicas y términos polinomiales

Dicos = pd.get\_dummies(mathfinalgrade[vars\_cat\_mod], drop\_first = True)

Y = mathfinalgrade.G3

for i in range(1,4):

  Xi = PolynomialFeatures(i).fit\_transform(mathfinalgrade[vars\_cuant\_mod])

  Xi = pd.DataFrame(Xi, index = mathfinalgrade.index.values)

  Xi = pd.concat([Dicos, Xi], axis=1)

  print('Modelo con dicotómicas y polinomial de grado', i)

  modelo2(Xi, Y)

#Comparamos los modelos con dicotómicas y polinomial con diferentes grados.

#Podemos observar que el mejor MSE es del modelo regresion ridge, de grado 3 (MSE= 4.1397)

Modelo con dicotómicas y polinomial de grado 1

reg lineal: -4.288584201094514

hiperparam ridge: {'alpha': 1}

reg ridge: -6.606779530650146

hiperparam lasso: {'alpha': 1}

reg lasso: -21.256092773593977

Modelo con dicotómicas y polinomial de grado 2

reg lineal: -1.3239689819568376e+19

hiperparam ridge: {'alpha': 1}

reg ridge: -4.249476569920891

hiperparam lasso: {'alpha': 1}

reg lasso: -21.256092773593977

Modelo con dicotómicas y polinomial de grado 3

reg lineal: -233.80694942171854

hiperparam ridge: {'alpha': 1}

reg ridge: -4.139744002577247

hiperparam lasso: {'alpha': 1}

reg lasso: -21.256092773593977