INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

MAESTRÍA EN CIENCIA DE DATOS

OPTIMIZACIÓN CONVEXA (P2022_MCD3395A)

PROYECTO DE APLICACIÓN DE LA MATERIA”

Ashwin Bhat

Carlos Caloca Gómez

Daniel Lagunas Barba

Leonardo Razo Islas

Antonio Sepúlveda Angulo

Contents

Marco Teórico

Modelos Lineales

A lo largo del Proyecto se utilizaron distintos modelos lineales enfocados a la regresión, donde el valor objetivo se espera ser una combinación lineal de las diversas características. En notación matemática el valor a predecir (y) sería tal y como lo siguiente:

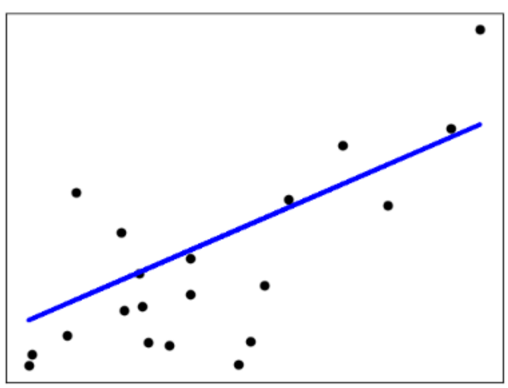
$$\hat{y}(w, x) = w_0 + w_1 x_1 + \ldots + w_p x_p$$

El valor w, será asignado al aspecto de coeficientes (*coef_*) y el intercepto será w_0, en las funciones a programar.

Mínimos Cuadrados Ordinarios

El modelo ajusta de acuerdo con los coeficientes del vector *w* con el fin de minimizar la suma de cuadrados de los residuales entre los valores observados de la base de datos con los valores predichos por una aproximación lineal. Matemáticamente lo pudiéramos representar de la siguiente manera:

$$\min_{w} || Xw - y||_2^2$$



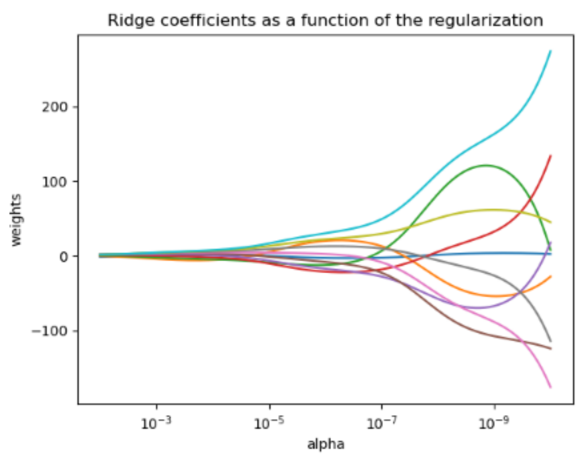
Las estimaciones de los coeficientes están basadas en la independencia de las características. Cuando las características están correlacionadas y tienen dependencia lineal, como consecuencia los valores predichos son altamente sensibles a errores aleatorios en los valores observados, generando una varianza mayor.

Regularización Ridge

Este modelo abarca algunos problemas de *Mínimos Cuadrados* al penalizar en el tamaño de los coeficientes. Los coeficientes Ridge minimizan la suma de cuadrados residuales penalizada.

$$\min_{w} ||Xw - y||_2^2 + \alpha||w||_2^2$$

El parámetro alfa mayor o igual a 0 controla la magnitud del encogimiento. A mayor valor de alfa, mayor cantidad de encogimiento y por ende los coeficientes se vuelven más robustos a colinealidad.



Regularización Lasso

Estima los coeficientes de dispersión, es de gran utilidad por su tendencia a preferir soluciones con menos coeficientes con valor 0, reduciendo de manera efectiva el número de características. La función objetivo para minimizar es la siguiente:

$$\min_{w} \frac{1}{2n_{\text{samples}}} ||Xw - y||_2^2 + \alpha ||w||_1$$

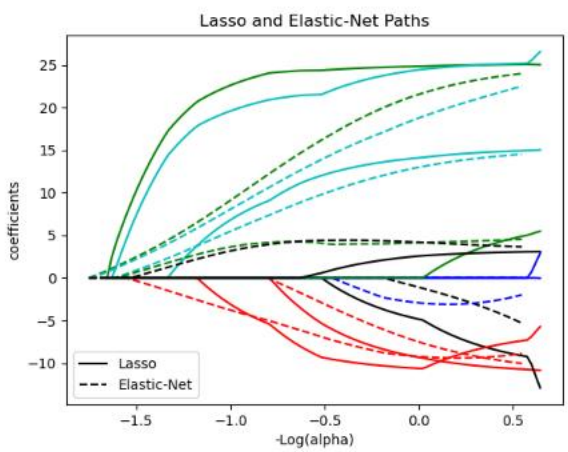
Elastic-Net

Es un modelo lineal entrenado con ℓ_1 y ℓ_2 coeficientes de regularización. Esta combinación nos permite aprender un modelo dispersión con algunos valores en cero con *Lasso* y aún así mantener las propiedades de regularización de *Ridge*.

La *Elastic-Net* es útil cuando tenemos muchas características que están correlacionadas entre sí.

La función objetivo a minimizar es la siguiente:

$$\min_{w} \frac{1}{2n_{\text{samples}}} ||Xw - y||_2^2 + \alpha\rho ||w||_1 + \frac{\alpha(1-\rho)}{2} ||w||_2^2$$



Comparativo entre modelos

Los modelos de Ridge y Lasso si bien hasta cierto punto son algo similares, tienen ciertas distinciones que es importante dejar en claro ya que posteriormente se seleccionará la mejor opción dado la naturaleza de los datos y del proyecto.

- *Lasso* obtiene que algunos coeficientes sean exactamente cero, por lo que realiza una selección de predictores. Por otro lado, *Ridge* no excluye ninguno. Esto se considera una ventaja ya que no todos los predictores son importantes y de igual manera estaríamos optimizando los recursos computacionales.
- *Ridge* reduce la influencia de los predictores altamente correlacionados y de forma proporcional. Por otro lado, *Lasso* se inclina por seleccionar un predictor y darle todo el peso excluyendo a los demás predictores, haciendo que las soluciones de *Lasso* sean muy inestables si existe alta correlación.

Ejecución

Introducción

Los productores de carne de pollo tienen como objetivo, de manera muy general, convertir de manera más eficiente los granos que consumen sus aves en carne. Existen muchas métricas y variables que se evalúan durante el crecimiento del pollo como:

- Conversión alimenticia
- Ganancia de peso diario
- Mortandad
- Mortalidad
- Integridad Intestinal
- Integridad Respiratoria
- Incidencia de enfermades
- Peso final

Todas ellas estas relacionadas entre sí y son utilizadas para verificar como crecen las parvadas y cual es la rentabilidad del negocio. La métrica de Integridad Intestinal es utilizada de manera subclínica para indicar cual es la calidad con la cual el pollo está procesando el alimento y si presenta alguna lesión interna que nos ayuda a identificar problemas de nutrición, bioseguridad y salud. Para obtener está métrica se requiere hacer un muestreo de al menos 5 aves por edad por caseta por parvada y realizar una necropsia detallada donde se revisan alrededor de 60 partes de órganos del ave, ingresarlas a un software llamado HTS y te dará 2 resultados: Integridad Intestinal e Integridad Respiratoria.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.svm import SVR
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import *
```

```
df_db_1435 = pd.read_excel('c:\Proyect\db_1419_all.xlsx',sheet_name='Sheet1')
```

```
# Se imprime los primeros 5 registro de la tabla

df_db_1435.head(5)
```


```
# Se imprime los primeros 5 registro de la tabla

df_db_1435.head(5)
```


	Aflt	Customer	PostingMonth	PostingYear	Age	AgePhase	Breed	Bird	a	AB	...	SX	TDS	TH	THY	TK	TN	TRA	TW	WC	I2
0	MX	716	November	2016	14	2	Arbor Acres	1	0.0	0	...	0	0	0	0	0	0	0	0	0	94
1	MX	716	November	2016	14	2	Arbor Acres	2	0.0	0	...	0	0	0	0	0	0	0	0	0	100
2	MX	716	November	2016	14	2	Arbor Acres	5	0.0	0	...	0	0	0	0	0	0	0	0	0	100
3	MX	716	November	2016	14	2	Arbor Acres	3	0.0	0	...	0	0	0	0	0	0	0	0	0	92
4	MX	716	November	2016	14	2	Arbor Acres	4	0.0	0	...	0	0	0	0	0	0	0	0	0	100

5 rows × 73 columns

```
# Columnas

df_db_1435.columns
```

```
Index(['Aflt', 'Customer', 'PostingMonth', 'PostingYear', 'Age', 'AgePhase',
       'Breed', 'Bird', 'a', 'AB', 'AC', 'AP', 'Ars', 'b', 'BDM', 'BF', 'BL',
       'BOW', 'BOX', 'BSM', 'BTL', 'Bur', 'BUW', 'CCS', 'Cdv', 'CFM', 'CL',
       'Clr', 'CM', 'CS', 'DC', 'FH', 'FP', 'gAC', 'gBR', 'Giz', 'gMI', 'gMx',
       'gNX', 'gTN', 'GUL', 'GUW', 'HK', 'HY', 'IH', 'IP', 'IT', 'L', 'LE',
       'MC', 'ML', 'mMX', 'NE', 'OM', 'PL', 'PRV', 'RI', 'RKT', 'RW', 'RY',
       'SC', 'SH', 'SPW', 'SX', 'TDS', 'TH', 'THY', 'TK', 'TN', 'TRA', 'TW',
       'WC', 'I2'],
      dtype='object')
```

```
# Remover las columnas de integridad respratoria, para dejar exclusivamente las lesiones de Integridad Intestinal

df_db_1435.drop(['Ars','AP','TRA','RI'],axis=1,inplace=True)
```

```
df_db_1435.columns
```

```
Index(['Aflt', 'Customer', 'PostingMonth', 'PostingYear', 'Age', 'AgePhase',
       'Breed', 'Bird', 'a', 'AB', 'AC', 'b', 'BDM', 'BF', 'BL', 'BOW', 'BOX',
       'BSM', 'BTL', 'Bur', 'BUW', 'CCS', 'Cdv', 'CFM', 'CL', 'Clr', 'CM',
       'CS', 'DC', 'FH', 'FP', 'gAC', 'gBR', 'Giz', 'gMI', 'gMx', 'gNX', 'gTN',
       'GUL', 'GUW', 'HK', 'HY', 'IH', 'IP', 'IT', 'L', 'LE', 'MC', 'ML',
       'mMX', 'NE', 'OM', 'PL', 'PRV', 'RKT', 'RW', 'RY', 'SC', 'SH', 'SPW',
       'SX', 'TDS', 'TH', 'THY', 'TK', 'TN', 'TW', 'WC', 'I2'],
      dtype='object')
```

```python
# Se revisan los tipos de datos, solo para verificar que no estuvieran como strings

df_db_1435.dtypes
```

```
Aflt            object
Customer         int64
PostingMonth    object
PostingYear      int64
Age              int64
                 ...  
TK               int64
TN               int64
TW               int64
WC               int64
I2               int64
Length: 69, dtype: object
```

```python
# Se hace el objeto para entrenar el modelo de regresion multlinear y el split de test y train

lr_multple=linear_model.LinearRegression()
x_train,x_test,y_train,y_test = train_test_split(df_db_1435.iloc[:,8:-1],df_db_1435.iloc[:,-1],test_size=0.2)

lr_multple.fit(x_train,y_train)
```

```
LinearRegression()
```

```python
# Se genera la prediccion

y_predict=lr_multple.predict(x_test)
```

```python
print('valor de los coeficientes')
print(lr_multple.coef_,'\n')

print('valor del intercepto')
print(lr_multple.intercept_,'\n')

print('score del entrenamiento')
print(lr_multple.score(x_train,y_train),'\n')

print('score del test')
print(lr_multple.score(x_test,y_test))
```

```
valor de los coeficientes
[-1.98546621e-14  4.28303226e-14 -7.57865992e-14  4.66293670e-15
  6.97220059e-14 -1.77635684e-14 -1.00000000e+00 -2.22044605e-16
  1.11022302e-15  1.55431223e-15  1.53121960e-12  4.70734562e-14
 -7.10542736e-15  9.76996262e-15 -9.28146449e-14  5.32240918e-12
 -1.06581410e-14 -7.54951657e-15 -2.22044605e-15 -3.00000000e+00
  3.07864845e-13 -1.57207580e-13 -2.00000000e+00 -3.00000000e+00
 -1.33226763e-15 -3.00000000e+00 -2.00000000e+00 -5.00000000e+00
  3.10862447e-15 -2.00000000e+00 -5.55111512e-16  0.00000000e+00
 -2.89546165e-13 -5.00000000e+00 -1.00000000e+00 -9.63673585e-14
 -2.00000000e+00 -7.05879799e-13 -4.00790512e-14 -3.00000000e+00
 -2.00000000e+00 -3.55271368e-15 -5.00000000e+01 -8.31112956e-13
  0.00000000e+00 -3.00000000e+00  2.74891221e-13  0.00000000e+00
  9.32587341e-15  6.25055563e-14 -5.12478948e-13  0.00000000e+00
  7.54396545e-14  5.91748872e-14 -3.00000000e+00 -9.21485110e-15
 -3.00000000e+00  2.86881630e-13 -1.00000000e+00 -2.00000000e+00]

valor del intercepto
100.00000000000033

score del entrenamiento
1.0

score del test
1.0
```

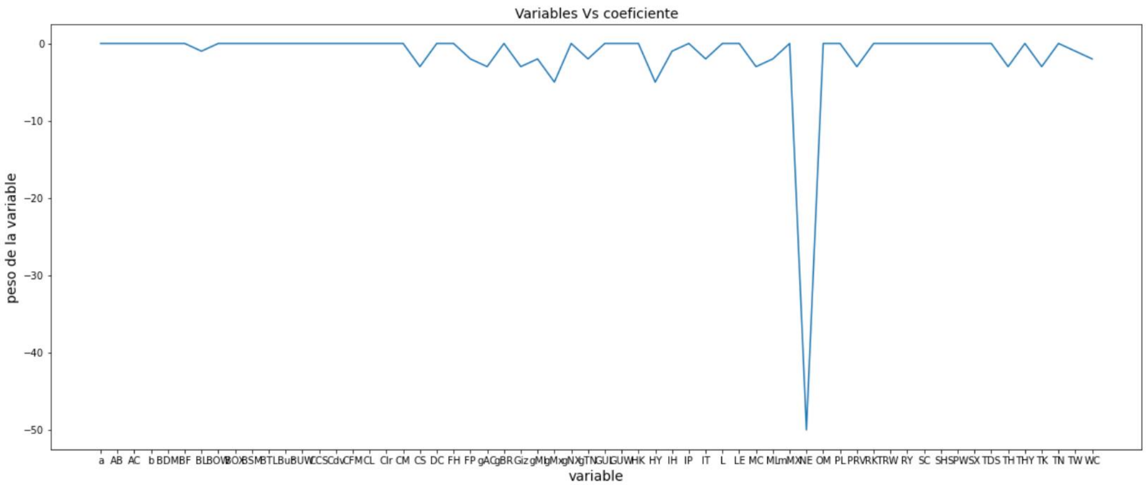
```python
# Valoracion con maximo daño

lr_multple.predict(np.array(df_db_1435.iloc[:,8:-1].max()).reshape(1,-1))
```

```
array([-40.])
```

```python
# Valoracion con minimo daño , lo cual hace sentido porque todos los pollos inician con calificacion de 100

lr_multple.predict(np.array(df_db_1435.iloc[:,8:-1].min()).reshape(1,-1))
```

```
array([100.])
```

```python
plt.rcParams["figure.figsize"] = (20,8)
plt.plot(df_db_1435.columns[8:-1],lr_multple.coef_)
plt.title('Variables Vs coeficiente', fontsize=14)
plt.xlabel('variable', fontsize=14)
plt.ylabel('peso de la variable', fontsize=14)
#plt.grid(True)
plt.show()
```




```
# Se buscan las variables con poco peso

col_name=np.array(df_db_1435.iloc[:,8:-1].columns.tolist())
var_less_weight = col_name[abs(lr_multple.coef_)< 0.01]
var_less_weight
```

```
array(['a', 'AB', 'AC', 'b', 'BDM', 'BF', 'BOW', 'BOX', 'BSM', 'BTL',
       'Bur', 'BUW', 'CCS', 'Cdv', 'CFM', 'CL', 'Clr', 'CM', 'DC', 'FH',
       'gBR', 'gNX', 'GUL', 'GUW', 'HK', 'IP', 'L', 'LE', 'mMX', 'OM',
       'PL', 'RKT', 'RW', 'RY', 'SC', 'SH', 'SPW', 'SX', 'TDS', 'THY',
       'TN'], dtype='<U3')
```

```
# Se remueven las variables con poco peso

df_db_1435.drop(var_less_weight,axis=1, inplace=True)
```

```
df_db_1435.columns
```

```
Index(['Aflt', 'Customer', 'PostingMonth', 'PostingYear', 'Age', 'AgePhase',
       'Breed', 'Bird', 'BL', 'CS', 'FP', 'gAC', 'Giz', 'gMI', 'gMx', 'gTN',
       'HY', 'IH', 'IT', 'MC', 'ML', 'NE', 'PRV', 'TH', 'TK', 'TW', 'WC',
       'I2'],
      dtype='object')
```

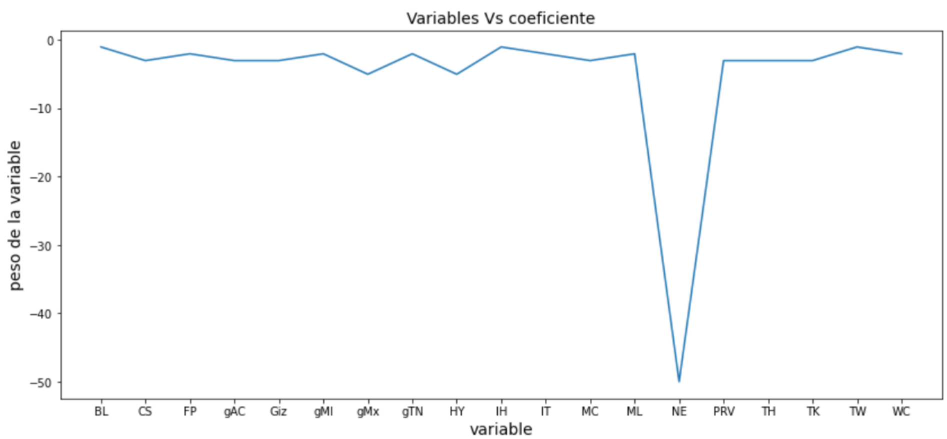
```
# Se vuelve hacer el entrenamiento del modelo y el split test train

x_train,x_test,y_train,y_test = train_test_split(df_db_1435.iloc[:,8:-1],df_db_1435.iloc[:,-1],test_size=0.2)

lr_multple.fit(x_train,y_train)
```

```
LinearRegression()
```

```
print('valor de los coeficientes')
print(lr_multple.coef_,'\n')

print('valor del intercepto')
print(lr_multple.intercept_,'\n')

print('score del entrenamiento')
print(lr_multple.score(x_train,y_train),'\n')

print('score del test')
print(lr_multple.score(x_test,y_test))
```

```
valor de los coeficientes
[ -1.  -3.  -2.  -3.  -3.  -2.  -5.  -2.  -5.  -1.  -2.  -3.  -2. -50.
  -3.  -3.  -3.  -1.  -2.]

valor del intercepto
100.0000000000001

score del entrenamiento
1.0

score del test
1.0
```

```
plt.rcParams["figure.figsize"] = (14,6)
plt.plot(df_db_1435.columns[8:-1],lr_multple.coef_)
plt.title('Variables Vs coeficiente', fontsize=14)
plt.xlabel('variable', fontsize=14)
plt.ylabel('peso de la variable', fontsize=14)
#plt.grid(True)
plt.show()
```




```
# Variables y Coeficientes

pd.DataFrame([df_db_1435.columns[8:-1],lr_multple.coef_])
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	BL	CS	FP	gAC	Giz	gMI	gMx	gTN	HY	IH	IT	MC	ML	NE	PRV	TH	TK	TW	WC
1	-1.0	-3.0	-2.0	-3.0	-3.0	-2.0	-5.0	-2.0	-5.0	-1.0	-2.0	-3.0	-2.0	-50.0	-3.0	-3.0	-3.0	-1.0	-2.0

```
# Errores 0_1
# Porque si no tiene erores el I2 sale diferente a 100

df_db_1435.iloc[:,8:][ (df_db_1435.iloc[:,8:-1].sum(axis=1)==0) & (df_db_1435.I2 == 100) ]
```

	BL	CS	FP	gAC	Giz	gMI	gMx	gTN	HY	IH	IT	MC	ML	NE	PRV	TH	TK	TW	WC	I2
1	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	100.0
2	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	100.0
4	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	100.0
10	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	100.0
27	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	100.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
28012	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	100.0
28044	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	100.0
28065	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	100.0
28066	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	100.0
28129	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	100.0

6057 rows × 20 columns

```
df_db_1435.iloc[:,8:][ (df_db_1435.iloc[:,8:-1].sum(axis=1)==0) & (df_db_1435.I2 != 100) ]
```

BL	CS	FP	gAC	Giz	gMI	gMx	gTN	HY	IH	IT	MC	ML	NE	PRV	TH	TK	TW	WC	I2

Lasso Regression

Get the data from Excel and Store the values in the dataframe

```
df_db_1435 = pd.read_excel(r'C:\Users\ASBHBHAT\Downloads\db_1419_all.xlsx',sheet_name='Sheet1')
```

Drop the unnecessary columns(Data Preprocessing)

```
df_db_1435.drop(['Ars','AP','TRA','RI'],axis=1,inplace=True)
df_db_1435.columns
```

```
Index(['Aflt', 'Customer', 'PostingMonth', 'PostingYear', 'Age', 'AgePhase',
       'Breed', 'Bird', 'a', 'AB', 'AC', 'b', 'BDM', 'BF', 'BL', 'BOW', 'BOX',
       'BSM', 'BTL', 'Bur', 'BUW', 'CCS', 'Cdv', 'CFM', 'CL', 'Clr', 'CM',
       'CS', 'DC', 'FH', 'FP', 'gAC', 'gBR', 'Giz', 'gMI', 'gMx', 'gNX', 'gTN',
       'GUL', 'GUW', 'HK', 'HY', 'IH', 'IP', 'IT', 'L', 'LE', 'MC', 'ML',
       'mMX', 'NE', 'OM', 'PL', 'PRV', 'RKT', 'RW', 'RY', 'SC', 'SH', 'SPW',
       'SX', 'TDS', 'TH', 'THY', 'TK', 'TN', 'TW', 'WC', 'I2'],
      dtype='object')
```

Split the data into train and test data set in the ratio of 4:1

```
x_train,x_test,y_train,y_test=train_test_split(df_db_1435.iloc[:,8:-1],df_db_1435.iloc[:,-1],test_size=0.2)
```

Sklearn SVR Module

we are using SVR module with Linear Kernel with Hyperparameters C and Epsilon. The Hyperparameters helps in tuning of the model to give us best result. we used GridsearchCV against different C and epsilon values, however due to quantity of data, we were not getting the result in time. Hence we manually checked using different C and epsilon value and we got a R2 score of 0.996 for C=1.0 and epsilon=0.3 which states that it is a good model.

```
#param_grid = {'kernel': ['linear'], 'C':[1, 1.5, 2, 10],'epsilon':[0.2,0.5,0.3]}
svr = svm.SVR()
regr = make_pipeline(StandardScaler(), SVR(kernel='linear', C=1.0, epsilon=0.3))
#regr = GridSearchCV(svr, param_grid)
regr.fit(x_train, y_train)
y_pred = regr.predict(x_test)
print("score:", regr.score(x_test, y_test))
```

```
score: 0.9967708218779135
```

Regression Metrics

We calculate some of the Linear Regression Metrics like MAE, MSE and RMSE. The ideal values of these metrics would be 0 indicating that there is no error.We get Error $<$ 0.3 with R^2 score of 0.99 indicating that we have a good model.

MSE mean squared error which calculate the square of error and is given by below formula.

$$MSE = \frac{1}{n_{samples}} \sum_{i=0}^{i=n_{samples}-1} (y_i - \bar{y}_i)^2$$

MAE is robust to outliers as it calculates the median of errors and is given by below formula.

$$MAE = median(\mid y_1 - \bar{y}_1 \mid, \ldots, \mid y_n - \bar{y}_n \mid)$$

RMSE is Root mean squared error which calculate the root of square of error and is given by below formula.

$$RMSE = \frac{1}{n_{samples}} \sum_{i=0}^{i=n_{samples}-1} |y_i - \bar{y}_i|$$

```
print("RMSE:", mean_squared_error(y_test, y_pred, squared=False))
print("MAE :", mean_absolute_error(y_test, y_pred))
print("MSE :", mean_squared_error(y_test, y_pred))
print("R2 score:", r2_score(y_test, y_pred))
```

```
RMSE: 0.29988341846397093
MAE : 0.2998368082337685
MSE : 0.08993006466963711
R2 score: 0.9967708218779135
```

```
print("RMSE:", mean_squared_error(y_test, y_pred, squared=False))
print("MAE :", mean_absolute_error(y_test, y_pred))
print("MSE :", mean_squared_error(y_test, y_pred))
print("R2 score:", r2_score(y_test, y_pred))
```

```
RMSE: 0.29988341846397093
MAE : 0.2998368082337685
MSE : 0.08993006466963711
R2 score: 0.9967708218779135
```

Lasso Regularisation

We are going to check the model using Lasso Regularisation with hyperparameter $alpha=0.03$. Lasso L1 Regularisation helps in reducing number of Regression coefficients for the columns that are not needed in model. we get a very good R^2 score for train and test using Lasso Regularisation.

```
reg = linear_model.Lasso(alpha=0.001)
reg.fit(x_train,y_train )

print(reg.coef_)
print(reg.intercept_)
print("Regression with Lasso Regularisation train score:",reg.score(x_train,y_train))
print("Regression with Lasso Regularisation test score:",reg.score(x_test,y_test))
```

```
[-0.00000000e+00  0.00000000e+00 -0.00000000e+00 -1.78020732e-04
 -0.00000000e+00 -0.00000000e+00 -9.79042335e-01  4.87036936e-07
  0.00000000e+00 -8.97641575e-05 -0.00000000e+00 -0.00000000e+00
  0.00000000e+00  0.00000000e+00 -1.50681871e-03  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00 -2.99599379e+00
  0.00000000e+00 -0.00000000e+00 -1.98693613e+00 -2.99519244e+00
  0.00000000e+00 -2.99995230e+00 -0.00000000e+00 -4.99699613e+00
  0.00000000e+00 -1.99630589e+00  0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -4.99650507e+00 -9.83974481e-01 -0.00000000e+00
 -1.98872442e+00 -0.00000000e+00 -0.00000000e+00 -2.99749639e+00
 -2.00327842e+00  0.00000000e+00 -3.85098958e+01 -0.00000000e+00
  0.00000000e+00 -2.99542783e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00 -0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00 -2.98163063e+00 -0.00000000e+00
 -2.97864906e+00  0.00000000e+00  0.00000000e+00 -1.99059583e+00]
99.9893323042291
Regression with Lasso Regularisation train score: 0.999557716927079
Regression with Lasso Regularisation test score: 0.997511690909845
```

Removing all the useless columns

we are able to reduce the number of parameters from 60 to 19 using L1 Regularisation and create a Regression Model from that.

```
col_index = [index for index,x in enumerate(reg.coef_) if x.round(3) != 0]
df_db_1435.columns[col_index]
```

```
Index(['Breed', 'BL', 'Bur', 'Cdv', 'CFM', 'Clr', 'CS', 'FH', 'Giz', 'gMI',
       'gNX', 'GUW', 'HK', 'IH', 'L', 'RKT', 'RY', 'SPW'],
      dtype='object')
```

Lasso Regularised Model

```
model = ''
for i in col_index:
    model = model + str(round(reg.coef_[i],3)) + str(df_db_1435.columns[i]) + ' '

print()
print()
print("The model based on Lasso regularisation is given as:")
print(model, ' + ' ,str(reg.intercept_))
```

```
The model based on Lasso regularisation is given as:
-0.979Breed -0.002BL -2.996Bur -1.987Cdv -2.995CFM -3.0Clr -4.997CS -1.996FH -4.997Giz -0.984gMI -1.989gNX -2.997GUW -2.003HK -
38.51IH -2.995L -2.982RKT -2.979RY -1.991SPW   +  99.9893323042291
```

Calculating y_pred values using test data

```
y_pred = np.matmul(x_test.iloc[:,col_index], reg.coef_[col_index].reshape(len(col_index),1)) +
                    reg.intercept_*np.ones((x_test.shape[0],1))
```

Regression Metrics

we get RMSE as 0.26 and MSE as 0.069 which is good using now only the useful columns we get from Lasso Regularization.

```
print("RMSE:", mean_squared_error(y_test, y_pred, squared=False))
print("MAE :", mean_absolute_error(y_test, y_pred))
print("MSE :", mean_squared_error(y_test, y_pred))
print("R2 score:", r2_score(y_test, y_pred))
```

```
RMSE: 0.26325499357655935
MAE : 0.015363633872823738
MSE : 0.0693031916429943
R2 score: 0.9975114846067726
```

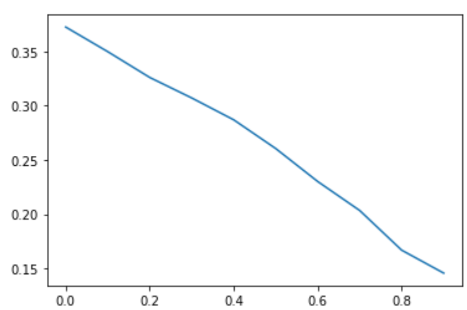
Elastic Net Regularization

Elastic Net Regularization uses the benefits of both Ridge and Lasso. It helps to remove the parameters which have very high correlation or collinearity between them. Using Elastic Net we get R^2 score of 0.28 suggesting it is not a good model.

```
reg = linear_model.ElasticNet(random_state=0)
reg.fit(x_train, y_train)

print(reg.coef_)
print(reg.intercept_)
print("Regression with Elastic Net Regularisation train score:",reg.score(x_train,y_train))
print("Regression with Elastic Net Regularisation test score:",reg.score(x_test,y_test))

score = []
for i in np.arange(0,1,0.1):
    reg = linear_model.ElasticNet(l1_ratio=i)
    reg.fit(x_train, y_train)
    score.append(reg.score(x_test, y_test))

plt.plot(np.arange(0,1,0.1), np.array(score))
plt.show()
```

```
[-0.00000000e+00  0.00000000e+00  0.00000000e+00 -4.34823125e-02
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -9.20849257e-04
  0.00000000e+00 -1.33161298e-01 -0.00000000e+00 -0.00000000e+00
  0.00000000e+00  0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  0.00000000e+00  0.00000000e+00 -0.00000000e+00 -4.25584808e-01
  0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -3.97396595e-01
  0.00000000e+00 -9.72077177e-01 -0.00000000e+00 -1.06869950e-01
  0.00000000e+00 -1.79415859e-01  0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -5.22703561e-01 -0.00000000e+00 -0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -5.00213762e-01
 -0.00000000e+00  0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  0.00000000e+00 -9.53973038e-02  0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00]
97.0832940661413
Regression with Elastic Net Regularisation train score: 0.2769557860027865
Regression with Elastic Net Regularisation test score: 0.2605991602258113
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:527: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 234085.04356778745, tolerance: 62.42135222405994
5
  model = cd_fast.enet_coordinate_descent(
```



Conclusiones

Podemos concluir que sí cumplimos con la hipótesis que teníamos al inicio de replicar la fórmula para estimar el índice de integridad intestinal (I2) a través de un modelo de regresión lineal, dando como mejor modelo primero el SVR, seguido de la regresión con regularización Lasso y por último el modelo simple de regresión lineal (aplicando Backward para la elección de las variables con base a su significancia estadística).

Con Lasso fue posible reducir el número de parámetros de 60 a 19 y obtuvimos un modelo de con 0.99 de significancia por lo tanto podemos concluir que con Lasso todavía es mejor.

Anexos

Binder

Git

Bibliografía

Rodrigo, J. A. (Noviembre de 2020). *Regularización, Ridge, Lasso, y Elastic Net con Python.* Obtenido de Ciencia de Datos: https://www.cienciadedatos.net/documentos/py14-ridge-lasso-elastic-net-python.html

Scikit Learn. (s.f.). *Linear Models.* Obtenido de Scikit Learn: https://scikit-learn.org/stable/modules/linear_model.html#elastic-net

Sepulveda, A. (9 de Abril de 2022). I2 y su relación con lesiones intestinales del pollo. (A. B. Daniel Lagunas, Entrevistador)

StatQuest (Dirección). (2018). *Regularization Part 1: Ridge (L2) Regression* [Película].

StatQuest (Dirección). (2018). *Regularization Part 2: Lasso (L1) Regression* [Película].

StatQuest (Dirección). (2018). *Regularization Part 3: Elastic Net Regression* [Película].