Asignatura	Datos del alumno	Fecha
Machine Learning	**Apellidos:** Avilés Cahill	27/02/2024
	Nombre: Adán	

Laboratorio: Resolver un problema de regresión

Índice

Asignatura	Datos del alumno	Fecha
Machine Learning	**Apellidos:** Avilés Cahill	27/02/2024
	Nombre: Adán	

1. Introducción

En esta actividad, vamos a resolver un problema de regresión, creando primero un dataset ficticio.

2. Creación del dataset

El primer paso consiste en crear ese dataset, en el cual emplearemos el DNI para tener un conjunto de datos distintos. Para que los dataset sean comparables entre todos:

- Si el número de identidad tiene menos de 8 cifras replicaremos las primeras hasta obtener exactamente 8 (de forma manual)
- Si alguna de las cifras es menor que 2 la sustituiremos por ese número (automatizado)

. Crearemos una clase que nos transforme el DNI en la forma adecuada (fallando si no contiene 8 dígitos o si es un entero en vez de una string) y a partir de él, se creará el dataset. En el anexo 6.1 se puede encontrar todos los imports necesarios.

```
class DatasetGenerator:
    def __init__(self,
                 dni: str):
        self.dni = dni
        self._check_dni()
        self._prepare_dni()
        self.dataset = None

    def _check_dni(self):
        """
        Check if DNI is well defined
        Returns: The error

        """
        if not isinstance(self.dni, str):
```

```
            raise TypeError('DNI is not an string')
        if not self.dni.isnumeric():
            raise ValueError('DNI has characters that are not numbers')
        if len(self.dni) != 8:
            raise ValueError('DNI has not 8 digits')

    def _prepare_dni(self):
        """
        Change the 0 and 1 to 2
        Returns:

        """
        self.dni.replace('1', '2').replace('0', '2')

    def _create_dataset(self):
        X, y = make_regression(n_samples=200+10*int(self.dni[0]),  # 240
                               n_features=10+int(self.dni[1])+int(self.dni[2]),  # 25
                               n_informative=10+int(self.dni[1]),  # 14
                               n_targets=1,
                               bias=2,
                               noise=10*int(self.dni[3]),  # 17
                               shuffle=False,
                               random_state=int(self.dni))
        list_of_features = [
            f'feature_{number}' for number in
            range(0, 10+int(self.dni[1])+int(self.dni[2]))
        ]
        dataset = pd.DataFrame(X, columns=list_of_features)
```

```
        dataset['target'] = y
        self.dataset = dataset

    def get_clean_dni(self):
        return self.dni

    def get_dataset(self):
        if self.dataset is None:
            self._create_dataset()
        return self.dataset
```

Después simplemente hemos de importar la clase donde la necesitemos y ejecutar

```
dataset = DatasetGenerator(dni='48778094').get_dataset()
```

3. Preparación y EDA

3.1. Preparación

Separaremos en primer lugar el dataset en dos sub datasets, train y test. El train es el que usaremos durante el resto de la práctica mientras que el test lo dejaremos intacto hasta el final.

```
dataset_features = dataset.drop('target', axis=1)
target = dataset[['target']]
# Split the dataset into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(dataset_features,
                                                    target,
                                                    train_size=200,
                                                    random_state=48778094)
```

3.2. EDA

Hemos ahora de aplicar los métodos *info()* y *describe()* para entender un poco más nuestro dataset.

```
X_train.info()
# <class 'pandas.core.frame.DataFrame'>
# Int64Index: 200 entries, 180 to 118
# Data columns (total 25 columns):
#  #   Column      Non-Null Count  Dtype
# ---  ------      --------------  -----
#  0   feature_0   200 non-null    float64
#  1   feature_1   200 non-null    float64
#  2   feature_2   200 non-null    float64
#                     ...
```

Podemos observar que no tenemos missings en ninguna de las variables, y que ninguna es categórica, con lo cual no tendremos problema alguno, ni que imputar missings o tratar variables. Pasemos a ver la información que nos brinda describe()

```
X_train.describe()
#          feature_0   feature_1   feature_2  ...  feature_23  feature_24
# count  200.000000  200.000000  200.000000  ...  200.000000  200.000000
# mean     0.054070   -0.044148    0.089811  ...   -0.027000   -0.104768
# std      0.969202    0.952640    1.045291  ...    1.029931    0.978308
# min     -2.790527   -2.330851   -2.937990  ...   -2.312849   -2.667936
# 25%     -0.502178   -0.645493   -0.483369  ...   -0.705046   -0.756948
# 50%      0.120125    0.006028   -0.055528  ...    0.041570   -0.142016
# 75%      0.711147    0.599157    0.708519  ...    0.719215    0.639725
# max      3.037770    2.601626    2.743696  ...    3.310682    2.373275
```

Podemos notar que las features tienen todas una media cercana al 0 y una desviación típica de 1,

haciendo que en general los valores oteen entre -3 y 3. Por otro lado, el target no se comporta como ellas. En general, si las features tuvieran unos valores dispares, deberíamos estandarizarlas para evitar problemas (imaginemos pares de variables como peso en gramos y altura en metros), pero en este caso no será necesario ya que prácticamente la desviación típica es 1 la media 0.

3.3. Histograma y correlación

Realizamos el histograma ahora de las variables y el target.

```
df_train.hist(figsize = (16,18))
plt.show()
```

Que nos devuelve el histograma entero (que también hemos hecho variable por variable para ver su distribución), observando por ejemplo que la feature 12 está desplazada hacia la izquierda de la gráfica.

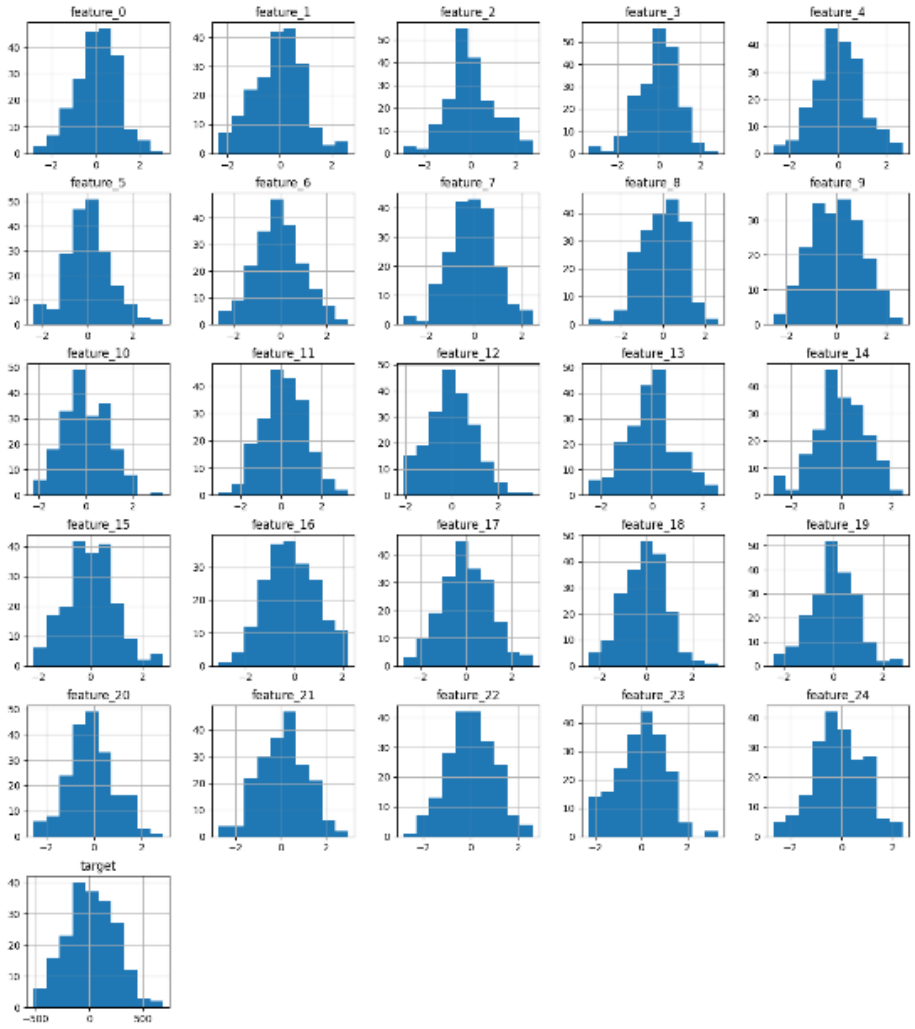

Figura 1: Histograma escalado

Para conocer un poco más de las variables, podemos ver su correlación entre ellas y con el target, utilizando

```
df_train=X_train.copy()
df_tain['target'] = y_train
corr = df_train.corr()
ax = sns.heatmap(
    corr,
```

```
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
)
plt.show()
```

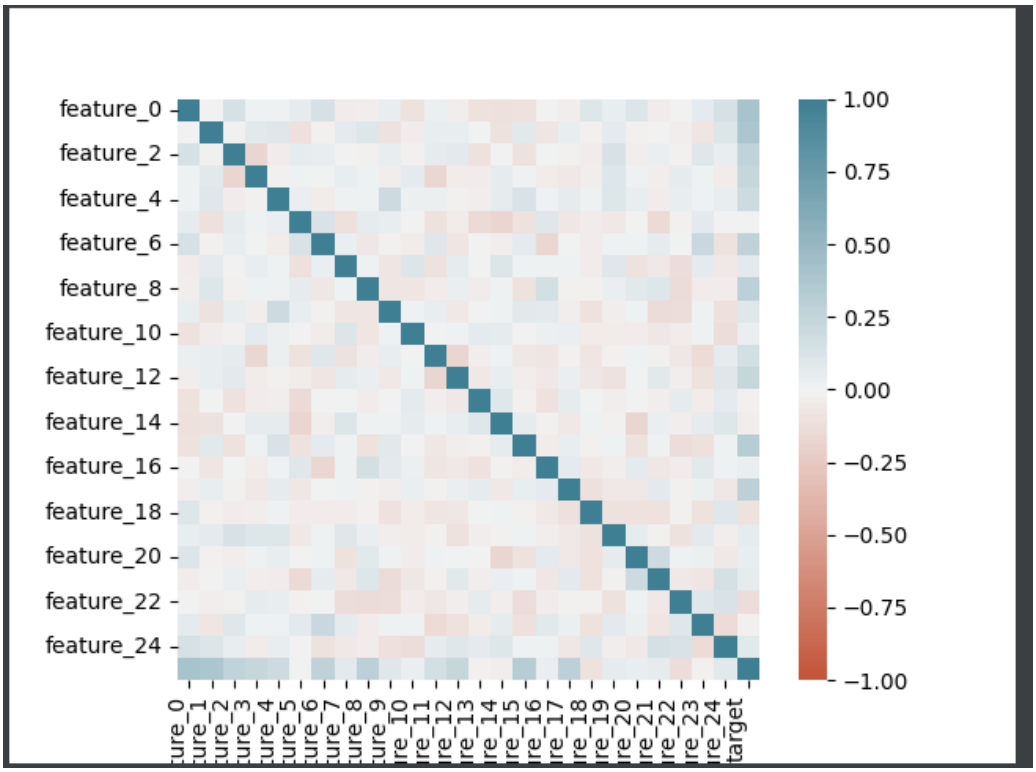

Figura 2: Correlaciones entre las fetaures y el target

Podemos observar que entre las features 15 y 24 y el target prácticamente no hay correlación. Esto nos puede dar una pista de qué variables van a ser las informativas, pero esto no tiene por qué ser exacto, es solo una idea.

4. Regresión lineal múltiple

Haremos ahora los tres modelos de regresión.

4.1. Regresión lineal múltiple sin tratamiento

Comprobaremos, haciendo una regresión simple y usando el paquete **statsmodels** si todas las variables son significativas.

```
X_train2 = sm.add_constant(X_train)
est = sm.OLS(y_train, X_train2) #Ordinary Least Squares
est2 = est.fit()
print(est2.summary())
```

Su resultado lo podemos ver en 6.2, observando que no todas las variables tienen un p.valor significativo, y por tanto podría ser que no añadan valor al modelo. Además, si nos fijamos en el coeficiente de determinación, podemos ver que es casi perfecto, superando el 0.9. Veamos si podemos mejorar el modelo aplicando un algoritmo stepwise para reducir las variables que no aporten nada.

4.2. Reducción de variables con step-wise

Como se nos pide que en cada paso quitemos una variable, haremos un backward step-wise. Usaremos *SKlearn* para nuestro cometido, pero no podremos aplicarlo directamente sobre *sm.OLS* ya que no está soportado (he creado una función que sí lo soporta y usa los p.valores, pero no estoy seguro de que esté funcionando adecuadamente así que no la he terminado de implementar)

```
 # Set n_features_to_select='auto' para eleccion automatica
 # Sabemos de antemano que son las informativas son n_features_to_select=14 en nuestro
 # Pero en un problema real no
sfs = SequentialFeatureSelector(LinearRegression(),
                                n_features_to_select=14)
sfs.fit_transform(X_train2, y_train)
```

```
final_features = sfs.get_feature_names_out()
# Podemos ahora quedarnos con estas featuers y comprobar con ols otra vez
est_stepwise = sm.OLS(y_train, X_train[sfs.get_feature_names_out()])
# ver ajuste----------------------------------------------------
est_stepwise_2 = est_stepwise.fit()
print(est_stepwise_2.summary())
```

En este caso, todas las variables tienen un p.valor significativo, el coeficiente de determinación ha disminuido un poco, pero no es algo que deba preocuparnos, pasar de un 0.95 a un 0.9 quitando, mientras disminuyes de 26 features a 13, no es algo preocupante. Significa que en esas 13 está la mayoría de la información.

4.3. Regresión con red elástica

Si jugamos con los distintos valores de r para la red elástica, podremos comprobar que los resultados varían bastante, peor para encontrar el óptimo usaremos la validación cruzada y un GridSearch.

```
 # Test de diferentes r
np.random.seed(48778094)
e_net = ElasticNetCV(cv=10,
                     l1_ratio=0.1)
e_net.fit(X_train.to_numpy(), y_train.values.ravel()) # Utilizamos ravel para evitar wa


e_net2 = ElasticNetCV(cv=10,
                     l1_ratio=0.95)
e_net2.fit(X_train.to_numpy(), y_train.values.ravel())

e_net3 = ElasticNetCV(cv=10,
                     l1_ratio=0.5)
e_net3.fit(X_train.to_numpy(), y_train.values.ravel())
```

Optimicemos ahora esta búsqueda con un GridSearch:

```
en_model = ElasticNet()
cv = RepeatedKFold(n_splits=10,
                   n_repeats=3,
                   random_state=48778094)
#Definimos el grid y sus parametros
grid = dict()
grid['alpha'] = [1e-5, 1e-3, 1e-1, 0.0, 1.0, 2.0, 3.0, 10.0, 100.0]
grid['l1_ratio'] = np.arange(0, 1, 0.01)
# define search
search = GridSearchCV(en_model,
                      grid,
                      cv=cv,
                      scoring='neg_mean_squared_error', #Usamos el mse para los errores
                      n_jobs=-1)
# Empezar la busqueda
results = search.fit(X_train, y_train)
print(f'MSE: {results.best_score_}')
print(f'Parametros: {results.best_params_}')
```

Y podemos ver que los parámetros optimos son $\alpha = 1$ y $r = 0{,}99$.

5. Comparativa de modelos

Computaremos el MSE de los tres modelos utilizando el test set que habíamos apartado anteriormente para comprobar cual obtiene un resultado mejor.

```
# Caso 1
# Añadimos el inercept al test para tratarlo igual que el train
```

```
X_test2 = sm.add_constant(X_test)
y_predict_ols = est2.predict(X_test2)

# Caso 2
# Nos quedamos con las variables necesarias
X_test_caso2 = X_test2[sfs.get_feature_names_out()]
y_predict_ols_step = est_stepwise_2.predict(X_test_caso2)

# Caso 3
y_predict_stepwise = results.predict(X_test)
```

Que nos da como resultados:

```
print(f'MSE for OLS: {mean_squared_error(y_test, y_predict_ols)}')
# MSE for OLS: 5915.2496749072825
print(f'MSE for OLS stepwise: {mean_squared_error(y_test, y_predict_ols_step)}')
# MSE for OLS stepwise : 6180.673002094224
print(f'MSE for ElasticNet: {mean_squared_error(y_test, y_predict_stepwise)}')
# MSE for ElasticNet: 5718.247443913208
```

Donde podemos observar que el mejor resultado lo obtenemos con ElasticNet. Igualmente, combinando estos algoritmos y tomando las variables adecuadas, podemos llegar a obtener un MSE mucho mejor.

6. Anexos: Código

6.1. Lista de imports

```
import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm

from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import ElasticNet, ElasticNetCV, LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import (train_test_split, GridSearchCV, RepeatedKFold)
from sklearn.preprocessing import StandardScaler

from Machine_Learning.practica_1.dataset_generator import DatasetGenerator
```

6.2. Resultado del summary

```
#                            OLS Regression Results
# ==============================================================================
# Dep. Variable:                 target   R-squared:                       0.914
# Model:                            OLS   Adj. R-squared:                  0.902
# Method:                 Least Squares   F-statistic:                     74.10
# Date:                Sat, 07 Jan 2023   Prob (F-statistic):           2.04e-79
# Time:                        14:33:32   Log-Likelihood:                -1121.8
# No. Observations:                 200   AIC:                             2296.
# Df Residuals:                     174   BIC:                             2381.
# Df Model:                          25
# Covariance Type:            nonrobust
# ==============================================================================
#                  coef    std err          t      P>|t|      [0.025      0.975]
# ------------------------------------------------------------------------------
# const          4.7299      5.345      0.885      0.377      -5.820      15.280
# feature_0     89.3163      5.574     16.023      0.000      78.314     100.318
# feature_1     64.8543      5.588     11.606      0.000      53.825      75.883
# feature_2     61.5356      5.126     12.005      0.000      51.419      71.653
# feature_3     76.1968      5.480     13.905      0.000      65.382      87.012
# feature_4     20.8159      5.404      3.852      0.000      10.150      31.482
# feature_5      9.3465      5.652      1.654      0.100      -1.809      20.502
# feature_6     47.2705      5.367      8.808      0.000      36.678      57.863
# feature_7     11.6897      5.622      2.079      0.039       0.594      22.785
# feature_8     72.9067      5.384     13.541      0.000      62.280      83.533
# feature_9     20.3513      5.361      3.796      0.000       9.769      30.933
# feature_10    16.4690      5.419      3.039      0.003       5.773      27.165
# feature_11    54.3916      5.118     10.627      0.000      44.290      64.493
```

```
# feature_12    66.7770    5.396    12.375    0.000    56.126    77.428
# feature_13    20.9228    5.326     3.929    0.000    10.412    31.434
# feature_14    -2.7940    5.504    -0.508    0.612   -13.658     8.070
# feature_15    90.4924    5.718    15.825    0.000    79.206   101.778
# feature_16    17.3160    5.219     3.318    0.001     7.016    27.616
# feature_17    60.6446    4.906    12.362    0.000    50.963    70.327
# feature_18    -0.9341    5.859    -0.159    0.874   -12.497    10.629
# feature_19    -8.1416    5.911    -1.377    0.170   -19.807     3.524
# feature_20     3.2295    5.609     0.576    0.566    -7.842    14.301
# feature_21     1.3802    5.317     0.260    0.795    -9.113    11.873
# feature_22    -3.5286    5.595    -0.631    0.529   -14.572     7.514
# feature_23     2.6554    5.367     0.495    0.621    -7.937    13.248
# feature_24     4.5158    5.711     0.791    0.430    -6.757    15.788
# ==============================================================================
# Omnibus:                        1.476   Durbin-Watson:                   2.126
# Prob(Omnibus):                  0.478   Jarque-Bera (JB):                1.459
# Skew:                          -0.123   Prob(JB):                        0.482
# Kurtosis:                       2.662   Cond. No.                         2.09
# ==============================================================================
```

7. Bibilografia

Referencias

[1] Apuntes de la asignatura de MachineLearning

[2] Documentación Oficial de Scikit-Learn

[3] Documentación Oficial de StatsModels