Open in Colab

Maestría en Inteligencia Artificial Aplicada

Curso: Inteligencia Artificial y Aprendizaje Automático

Tecnológico de Monterrey

Prof Luis Eduardo Falcón Morales

Actividad de la Semana 6

Árboles de decisión y bosque aleatorio.

Nombres y matrículas de los integrantes del equipo:

- Eduardo Aldair Ahumada García Jurado - A01422929
- Edgar Rodolfo Escobar Gomez - A01793900
- Walter André Hauri Rosales - A01794237
- Héctor Salvador Montañez Alvarez - A01332665
- Jaime Andres Palacios Campaña - A01794023

En cada sección deberás incluir todas las líneas de código necesarias para responder a cada uno de los ejercicios.

In [ ]:

```
# Incluye aquí todos módulos, librerías y paquetes que requieras.
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder, StandardScaler

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.ensemble import RandomForestClassifier

# from sklearn.model_selection import cross_val_score
from sklearn.model_selection import  cross_validate, learning_curve, validation_curv
from sklearn.model_selection import RepeatedStratifiedKFold
```

```
from sklearn.metrics import confusion_matrix, make_scorer

from graphviz import Source

# Adicionales--
from sklearn.preprocessing import power_transform
from sklearn.preprocessing import FunctionTransformer

# Usados para pruebas unitarias de código
from sklearn.datasets import make_moons, make_circles
```

Ejercicio-1.

Carga de datos

In [ ]:

```
# Cargamos la base de datos con los nombres de las columnnas traducidos a Inglés
# Este archivo se encuentra en nuestro repositorio github público:
# https://raw.githubusercontent.com/Edgar-IAH/IA-Grupo-45/main/SouthGermanCredit_Tra

datos = pd.read_csv("https://raw.githubusercontent.com/Edgar-IAH/IA-Grupo-45/main/So
### USAR LA SIGUIENTE LINEA UNICAMENTE CUANDO EL ARCHIVO DE DATOS SEA LOCAL
#datos = pd.read_csv("SouthGermanCredit_Translated.csv")

datos.describe()
```

Out[ ]:

	status	duration	credit_history	purpose	amount	savings	employment
count	1000.000000	1000.000000	1000.00000	1000.000000	1000.00000	1000.000000	10
mean	2.577000	20.903000	2.54500	2.828000	3271.24800	2.105000	
std	1.257638	12.058814	1.08312	2.744439	2822.75176	1.580023	
min	1.000000	4.000000	0.00000	0.000000	250.00000	1.000000	
25%	1.000000	12.000000	2.00000	1.000000	1365.50000	1.000000	
50%	2.000000	18.000000	2.00000	2.000000	2319.50000	1.000000	
75%	4.000000	24.000000	4.00000	3.000000	3972.25000	3.000000	
max	4.000000	72.000000	4.00000	10.000000	18424.00000	5.000000	

8 rows × 21 columns

Ejercicio-2.

Creación de la estructuras de datos de entrenamiento (_train) y prueba (_test)

In [ ]:

```
# imputacion por moda - Categóricos
datos_cat = [
    "status",
    "credit_history",
    "purpose",
    "savings",
    "personal_status_sex",
    "other_debtors",
    "other_installment_plans",
    "housing",
]

# imputacion por moda - Ordinales
datos_ord = [
    "employment_duration",
    "installment_rate",
    "present_residence",
    "property",
    "number_credits",
    "job",
]

# imputacion por media/mediana - Numéricos
datos_num = ["duration", "amount", "age"]

# imputacion por moda - Binarios
datos_bin = ["people_liable", "telephone", "foreign_worker"]
```

In [ ]:

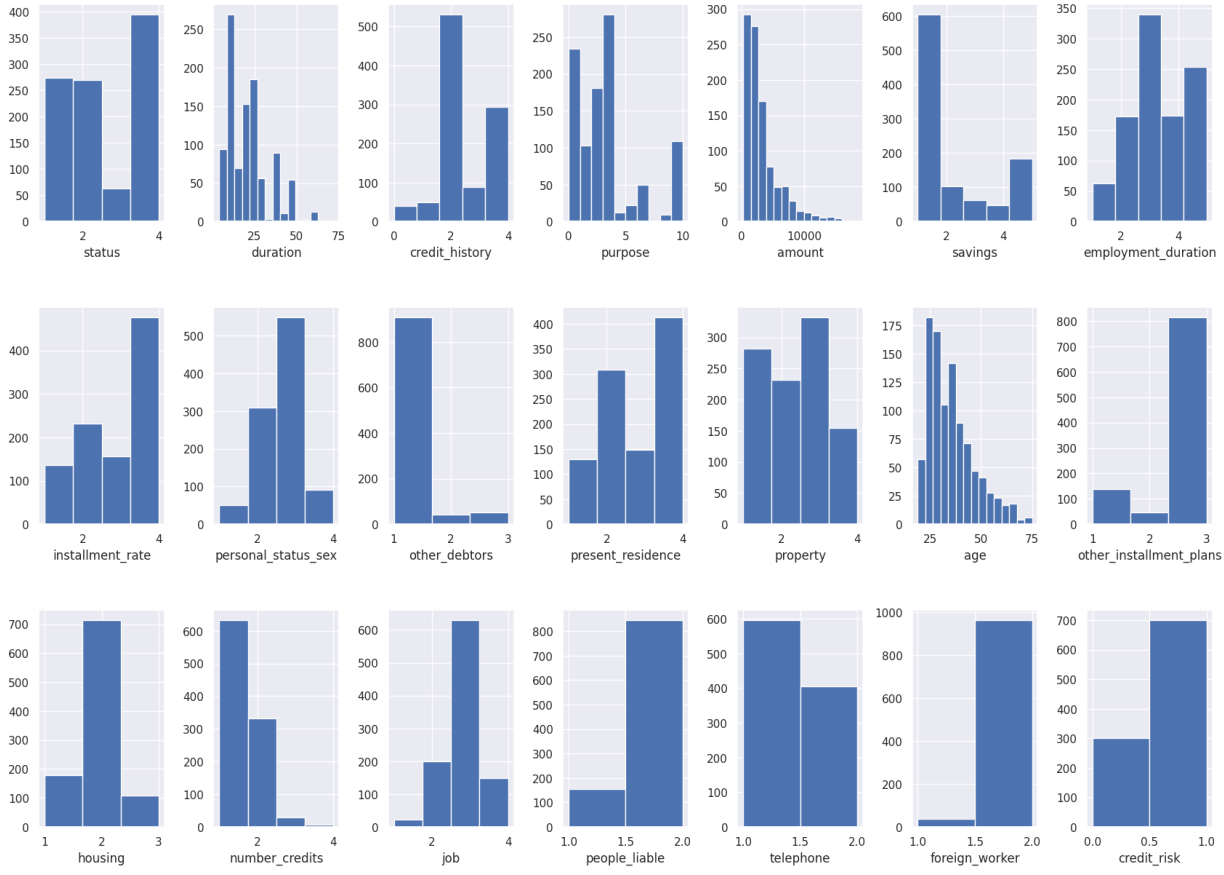
```
# Los datos de entrada son todas las columnas excepto la última
X = datos.iloc[:, :-1]
# La variable de salida se encuentra en la última columna
Y = datos.iloc[:, -1]

# Usamos la función train_test_split de la libreria sklearn
# Fijamos el generador de números aleatorios con el fin de que los conjuntos
# no cambien en cada corrida
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.15, random_state=45
)

# Chequeamos el tamaño de los conjuntos obtenidos
print(X_train.shape, ": dimensión de datos de entrada de entrenamiento y validación"
print(X_test.shape, ": dimensión de datos de entrada de prueba")
print(
    Y_train.shape, ": dimensión de variable de salida para entrenamiento y validació
)
print(Y_test.shape, ": dimensión de variable de salida para prueba")
```

```
(850, 20) : dimensión de datos de entrada de entrenamiento y validación
(150, 20) : dimensión de datos de entrada de prueba
(850,) : dimensión de variable de salida para entrenamiento y validación
(150,) : dimensión de variable de salida para prueba
```

In [ ]:

```
# En esta sección creamos los histogramas sin transformaciones
# para darnos una idea de las distribuciones

# Tamaño de la gráfica
sns.set(rc={"figure.figsize": (17, 12)})
```

```
# Crea el espacio de trabajo como una matriz de 3 x 7 = 21 variables
fig, axes = plt.subplots(3, 7)
# Espacio entre gráficas
fig.tight_layout(h_pad=5.0)

# Grafica los histogramas uno por uno
for k in range(0, 21):
    # Posición sequencial. Van de 1 a 21 en este caso (no 0)
    plt.subplot(3, 7, k + 1)
    # Número de valores diferentes.
    buckets = datos.iloc[:, k].nunique()
    # Si el número de valores es mayor de 10, declara el número de buckets como 15
    # si es <=10 usa ese número de buckets
    if buckets > 10:
        buckets = 15
    # Crea el histograma
    datos.iloc[:, k].hist(bins=buckets)
    # Nombra el eje X de acuerdo a la columna que está siendo visualizada
    plt.xlabel(datos.columns[k])
# Despliega la figura con las 21 gráficas
plt.show()
```



Ejercicio-3.

Primera approximación

Transformaciones:

- categórica (categorical),
- ordinal (discretized quantitative),
- numérica (quantitative), y
- binaria (binary).

In [ ]:
```
#Chequeamos si existen datos nulos. No esperamos ninguno ya que la base de datos
#que estamos usando ya ha sido limpiada anteriormente
datos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   status                   1000 non-null   int64
 1   duration                 1000 non-null   int64
 2   credit_history           1000 non-null   int64
 3   purpose                  1000 non-null   int64
 4   amount                   1000 non-null   int64
 5   savings                  1000 non-null   int64
 6   employment_duration      1000 non-null   int64
 7   installment_rate         1000 non-null   int64
 8   personal_status_sex      1000 non-null   int64
 9   other_debtors            1000 non-null   int64
 10  present_residence        1000 non-null   int64
 11  property                 1000 non-null   int64
 12  age                      1000 non-null   int64
 13  other_installment_plans  1000 non-null   int64
 14  housing                  1000 non-null   int64
 15  number_credits           1000 non-null   int64
 16  job                      1000 non-null   int64
 17  people_liable            1000 non-null   int64
 18  telephone                1000 non-null   int64
 19  foreign_worker           1000 non-null   int64
 20  credit_risk              1000 non-null   int64
dtypes: int64(21)
memory usage: 164.2 KB
```

In [ ]:
```
datos
```

Out[ ]:

	status	duration	credit_history	purpose	amount	savings	employment_duration	installment_
0	1	18	4	2	1049	1	2	
1	1	9	4	0	2799	1	3	
2	2	12	2	9	841	2	4	
3	1	12	4	0	2122	1	3	
4	1	12	4	0	2171	1	3	
...	...	...	...	...	...	...	...	
995	1	24	2	3	1987	1	3	
996	1	24	2	0	2303	1	5	

	status	duration	credit_history	purpose	amount	savings	employment_duration	installment_
997	4	21	4	0	12680	5	5	
998	2	12	2	3	6468	5	1	
999	1	30	2	2	6350	5	5	

1000 rows × 21 columns

In [ ]:
```
for c in datos_cat+datos_bin:
    print(datos[c].value_counts())
```

```
4    394
1    274
2    269
3     63
Name: status, dtype: int64
2    530
4    293
3     88
1     49
0     40
Name: credit_history, dtype: int64
3     280
0     234
2     181
1     103
9      97
6      50
5      22
10     12
4      12
8       9
Name: purpose, dtype: int64
1    603
5    183
2    103
3     63
4     48
Name: savings, dtype: int64
3    548
2    310
4     92
1     50
Name: personal_status_sex, dtype: int64
1    907
3     52
2     41
Name: other_debtors, dtype: int64
3    814
1    139
2     47
Name: other_installment_plans, dtype: int64
2    714
1    179
3    107
```

```
Name: housing, dtype: int64
2    845
1    155
Name: people_liable, dtype: int64
1    596
2    404
Name: telephone, dtype: int64
2    963
1     37
Name: foreign_worker, dtype: int64
```

In [ ]:
```
datos[datos_bin]
```

Out[ ]:

	people_liable	telephone	foreign_worker
0	2	1	2
1	1	1	2
2	2	1	2
3	1	1	1
4	2	1	1
...	...	...	...
995	1	1	2
996	2	1	2
997	2	2	2
998	2	2	2
999	2	1	2

1000 rows × 3 columns

In [ ]:
```
# Observación de los valores únicos en todas las columnas.
# Comparamos con los valores registrados en el archivo fuente "codetable.txt"
# con el fin de asegurarnos de que no haya valores en los datos que no estén
# documentados en el mencionado archivo fuente.
# Resultado negativo, lo cual es bueno. Todos los valores usados en los datos
# están referenciados en la en archivo fuente "codetable.txt".
for v in datos.columns:
    print(v)
    print(datos[v].unique())
    print("-----")
```

```
status
[1 2 4 3]
-----
duration
[18  9 12 10  8  6 24 11 30 48 36 15 42 21 27 33 28  4 47 14 39 60  5 22
 54 13 16  7 20 26 45 72 40]
-----
credit_history
[4 2 3 0 1]
-----
```

```
purpose
[ 2  0  9  3  1 10  5  4  6  8]
-----
amount
[ 1049  2799   841  2122  2171  2241  3398  1361  1098  3758  3905  6187
  1957  7582  1936  2647  3939  3213  2337  7228  3676  3124  2384  1424
  4716  4771   652  1154  3556  4796  3017  3535  6614  1376  1721   860
  1495  1934  3378  3868   996  1755  1028  2825  1239  1216  1258  1864
  1474  1382   640  3919  1224  2331  6313   385  1655  1053  3160  3079
  1163  2679  3578 10875  1344  1237  3077  2284  1567  2032  2745  1867
  2299   929  3399  2030  3275  1940  1602  1979  2022  3342  5866  2360
  1520  3651  2346  4454   666  1965  1995  2991  4221  1364  6361  4526
  3573  4455  2136  5954  3777   806  4712  7432  1851  1393  1412  1473
  1533  2012  3959   428  2366   763  3976  6260  1919  2603   936  3062
  4795  5842  2063  1459  1213  5103   874  2978  1820  2872  1925  2515
  2116  1453  1543  1318  2325   932  3148  3835  3832  5084  2406  2394
  2476  2964  1262  1542  1743   409  8858  3512  1158  2684  1498  6416
  3617  1291  1275  3972  3343   392  2134  5771  2788  5848  1228  1297
  1552  1963  3235  4139  1804  1950 12749  1236  1055  8072  2831  1449
  5742  2390  3430  2273  2923  1901  3711  8487  2255  7253  6761  1817
  2141  3609  2333  7824  1445  7721  3763  4439  1107  1444 12169  2753
  1494  2828  2483  1299  1549  3949  2901   709 10722  1287  3656  4679
  8613  2659  1516  4380   802  1572  3566  1278   426  8588  3857   685
  1603   601  2569  1316 10366  1568   629  1750  3488  1800  4151  2631
  5248  2899  6204   804  3595  5711  2687  3643  2146  2315  3448  2708
  1313  1493  2675  2118   909  1569  7678   660  2835  2670  3447  3568
  3652  3660  1126   683  2251  4675  2353  3357   672   338  2697  2507
  1478  3565  2221  1898   960  8133  2301   983  2320   339  5152  3749
  3074   745  1469  1374   783  2606  9436   930  2751   250  1201   662
  1300  1559  3016  1360  1204  1597  2073  2142  2132  1546  1418  1343
  2662  6070  1927  2404  1554  1283   717  1747  1288  1038  2848  1413
  3632  3229  3577   682  1924   727   781  2121   701  2069  1525  7629
  3499  1346 10477  2924  1231  1961  5045  1255  1858  1221  1388  2279
  2759  1410  1403  3021  6568  2578  7758   343  1591  3416  1108  5965
  1514  6742  3650  3599 13756   276  4041   458   918  7393  1225  2812
  3029  1480  1047  1471  5511  1206  6403   707  1503  6078  2528  1037
  1352  3181  4594  5381  4657  1391  1913  7166  1409   976  2375   522
  2743  5804  1169   776  1322  1175  2133  1829 11760  1501  1200  3195
  4530  1555  2326  1887  1264   846  1532   935  2442  3590  2288  5117
 14179  1386   618  1574   700   886  4686   790   766  2212  7308  5743
  3973  7418  2629  1941  2445  6468  7374  3812  4006  7472  2028  5324
  2323  1984   999  7409  2186  4473   937  3422  3105  2748  3872  5190
  3001  3863  5801  1592  1185  3780  3612  1076  3527  2051  3331  3104
  2611  1311  2108  4042   926  1680  1249  2463  1595  2058  7814  1740
  1240  6842  5150  1203  2080  1538  3878  3186  2896  6967  1819  5943
  7127  3349 10974   518  1860  9566  2930  1505  2238  2197  1881  1880
  2389  1967  3380  1455   730  3244  1670  3979  1922  1295  1544   907
  1715  1347  1007  1402  2002  2096  1101   894  1577  2764  8358  5433
  3485  3850  7408  1377  4272  1553  9857   362  1935 10222  1330  9055
  7966  3496  6948 12204  3446   684  4281  7174  2359  3621   741  7865
  2910  5302  3620  3509  1657  1164  6229  1193  4583  5371   708   571
  2522  5179  8229  1289  2712   975  1050   609  4788  3069   836  2577
  1620  1845  6579  1893 10623  2249  3108   958  9277  6314  1526  6615
  1872  2859  1582  1238  1433  7882  4169  3249  3149  2246  1797  2957
  2348  6289  6419  6143 15857  2223  7238  2503  2622  4351   368   754
  2424  6681  2427   753  2576   590  1414  1103   585  1068   713  1092
  2329   882   866  2415  2101  1301  1113   760   625  1323  1138  1795
  2728   484  1048  1155  7057  1537  2214  1585  1521  3990  3049  1282
 10144  1168   454  3594  1768 15653  2247  4576  8335  5800  8471  3622
  2181  7685  6110  3757  3394  6304  1244  3518  2613  7476  4591  5595
```

```
  6224  1905  2993  8947  4020  2779  2782  1884 11054  9157  9283  6527
  3368  2511  5493  1338  1082  1149  1308  6148  1736  3059  2996  7596
  4811  1766  2760  5507  1199  2892  2862   654  1136  4113 14555   950
  2150  2820  3060  2600  5003  6288  2538  4933  1530  1437  1823  1422
  1217  9271  2145  1842  4297  3384  1245  4623  8386  1024 14318   433
  2149  2397   931  1512  4241  4736  1778  2327  6872   795  1908  1953
  2864  2319   915   947  1381  1285  1371  1042   900  1207  2278  6836
  3345  1198 15672  7297  1943  3190  5129  1808   759  1980 10961  6887
  1938  1835  1659  1209  3844  4843   639  5951  3804  4463  7980  4210
  4611 11560  4165  4057  6458  1977  1928  1123 11328 11938  2520 14782
  2671 12612  3031   626  3931  2302  3965  3914  4308  1534  2775  5998
  1271  9398   951  1355  3051  7855  9572  1837  4249  5234  6758  1366
  1358  2473  1337  7763  6560  3123  8065  2439  9034 14027  9629  1484
  1131  2064 12976  2580  2570  3915  1309  4817  2579  2225  4153  3114
  2124  1333  7119  4870   691  4370  2746  4110  2462  2969  4605  6331
  3552   697  1442  5293  3414  2039  3161   902 10297 14421  1056  1274
  1223  1372  2625  2235   959   884  1246  8086 10127   888   719 12389
  6850  2210  7485   797  4746   939  1188 11590  1190  2767  3441  4280
  3092  1331 15945  3234  9960  8648  1345  1647  4844  8318  2100 11816
   448 11998 18424 14896  2762  3386  2169  5096  1882  6999  2292  8978
   674  2718   750 12579  7511  3966  6199  1987  2303 12680  6350]
-----
savings
[1 2 3 5 4]
-----
employment_duration
[2 3 4 1 5]
-----
installment_rate
[4 2 3 1]
-----
personal_status_sex
[2 3 4 1]
-----
other_debtors
[1 3 2]
-----
present_residence
[4 2 3 1]
-----
property
[2 1 3 4]
-----
age
[21 36 23 39 38 48 40 65 24 31 44 25 37 49 33 26 51 29 56 47 34 28 41 58
 61 30 63 27 45 43 52 22 60 32 35 42 59 54 64 46 74 50 20 55 53 19 57 66
 68 70 67 75 62]
-----
other_installment_plans
[3 1 2]
-----
housing
[1 2 3]
-----
number_credits
[1 2 3 4]
-----
job
[3 2 1 4]
-----
```

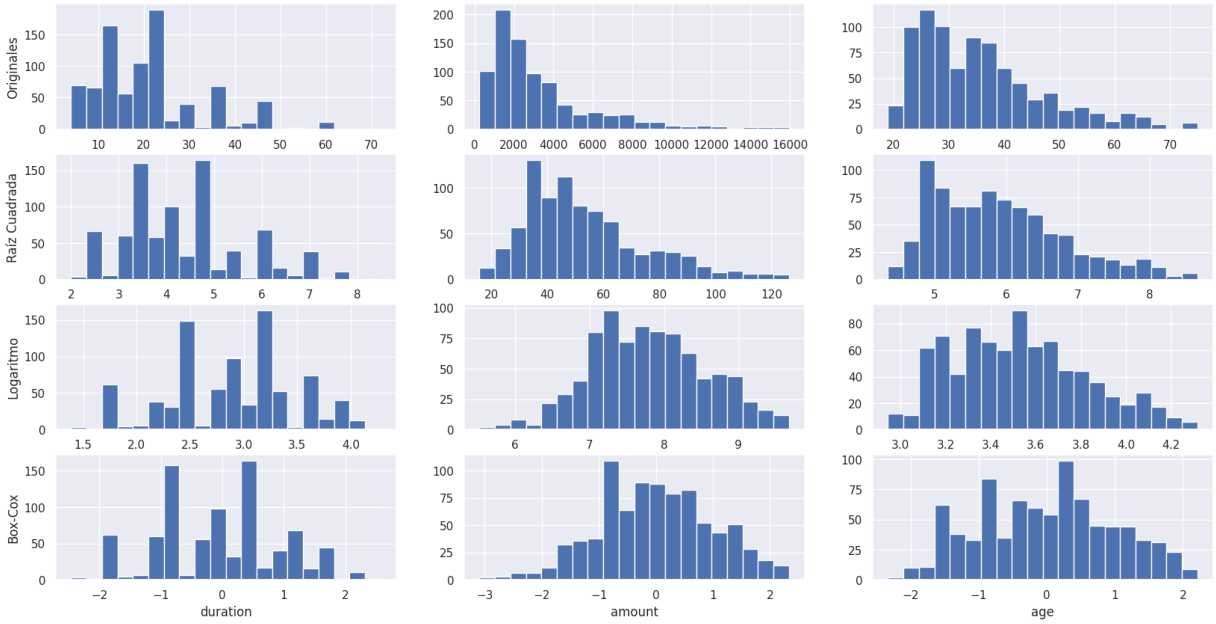
```
people_liable
[2 1]
-----
telephone
[1 2]
-----
foreign_worker
[2 1]
-----
credit_risk
[1 0]
-----
```

In [ ]:

```
# Mediante histogramas revisamos la distribución de frecuencias para encontrar
# cual daría la mejor simetria pos-transformación de las variable numéricas

sns.set(rc={"figure.figsize": (20, 10)})
fig, axes = plt.subplots(3, 3)

for k in range(0, 3):

    # Datos originales ----------------------------------------------------------
    plt.subplot(4, 3, k + 1)
    Transf0 = (
        X_train[datos_num].iloc[:, k].to_numpy().reshape(-1, 1)
    )  # Se asigna a Transf0 los valores de cada variable "k" sin transformar.
    plt.hist(
        Transf0, bins=20
    )  # Se agrega el comando para obtener el histograma de Transf0 con 20 barras (b
    plt.xlabel(datos_num[k])
    if k == 0:
        plt.ylabel("Originales")

    # Datos transformados con raíz cuadrada -----------------------------------
    plt.subplot(4, 3, k + 4)
    Transf1 = np.sqrt(
        Transf0
    )  # Se debe aplica la raíz cuadrada a los valores de cada variable "k" sin tran
    plt.hist(
        Transf1, bins=20
    )  # Se agrega el comando para obtener el histograma de Transf1 con 20 barras (b
    plt.xlabel(datos_num[k])
    if k == 0:
        plt.ylabel("Raíz Cuadrada")

    # Datos transformados con logaritmo natural -------------------------------
    plt.subplot(4, 3, k + 7)
    Transf2 = np.log(
        Transf0
    )  # Se aplica el logaritmo natural a los valores de cada variable "k" sin trans
    plt.hist(
        Transf2, bins=20
    )  # Se agrega el comando para obtener el histograma de Transf2 con 20 barras (b
    plt.xlabel(datos_num[k])
    if k == 0:
        plt.ylabel("Logaritmo")

    # Datos transformados con Box-Cox -----------------------------------------
    plt.subplot(4, 3, k + 10)
```

```python
    Transf4 = power_transform(
        (X_train[datos_num[k]]).values.reshape(-1, 1), method="box-cox"
    )  #  En esta línea se debe aplicar la transformación Box-Cox a los valores de c
    plt.hist(
        Transf4, bins=20
    )  #  En este línea agrega el comando para obtener el histograma de Transf4 con
    plt.xlabel(datos_num[k])
    if k == 0:
        plt.ylabel("Box-Cox")
```

```
/tmp/ipykernel_8484/2185687987.py:10: MatplotlibDeprecationWarning: Auto-removal of
overlapping axes is deprecated since 3.6 and will be removed two minor releases late
r; explicitly call ax.remove() as needed.
  plt.subplot(4, 3, k + 1)
/tmp/ipykernel_8484/2185687987.py:22: MatplotlibDeprecationWarning: Auto-removal of
overlapping axes is deprecated since 3.6 and will be removed two minor releases late
r; explicitly call ax.remove() as needed.
  plt.subplot(4, 3, k + 4)
/tmp/ipykernel_8484/2185687987.py:34: MatplotlibDeprecationWarning: Auto-removal of
overlapping axes is deprecated since 3.6 and will be removed two minor releases late
r; explicitly call ax.remove() as needed.
  plt.subplot(4, 3, k + 7)
/tmp/ipykernel_8484/2185687987.py:10: MatplotlibDeprecationWarning: Auto-removal of
overlapping axes is deprecated since 3.6 and will be removed two minor releases late
r; explicitly call ax.remove() as needed.
  plt.subplot(4, 3, k + 1)
/tmp/ipykernel_8484/2185687987.py:22: MatplotlibDeprecationWarning: Auto-removal of
overlapping axes is deprecated since 3.6 and will be removed two minor releases late
r; explicitly call ax.remove() as needed.
  plt.subplot(4, 3, k + 4)
/tmp/ipykernel_8484/2185687987.py:34: MatplotlibDeprecationWarning: Auto-removal of
overlapping axes is deprecated since 3.6 and will be removed two minor releases late
r; explicitly call ax.remove() as needed.
  plt.subplot(4, 3, k + 7)
/tmp/ipykernel_8484/2185687987.py:10: MatplotlibDeprecationWarning: Auto-removal of
overlapping axes is deprecated since 3.6 and will be removed two minor releases late
r; explicitly call ax.remove() as needed.
  plt.subplot(4, 3, k + 1)
/tmp/ipykernel_8484/2185687987.py:22: MatplotlibDeprecationWarning: Auto-removal of
overlapping axes is deprecated since 3.6 and will be removed two minor releases late
r; explicitly call ax.remove() as needed.
  plt.subplot(4, 3, k + 4)
/tmp/ipykernel_8484/2185687987.py:34: MatplotlibDeprecationWarning: Auto-removal of
overlapping axes is deprecated since 3.6 and will be removed two minor releases late
r; explicitly call ax.remove() as needed.
  plt.subplot(4, 3, k + 7)
```

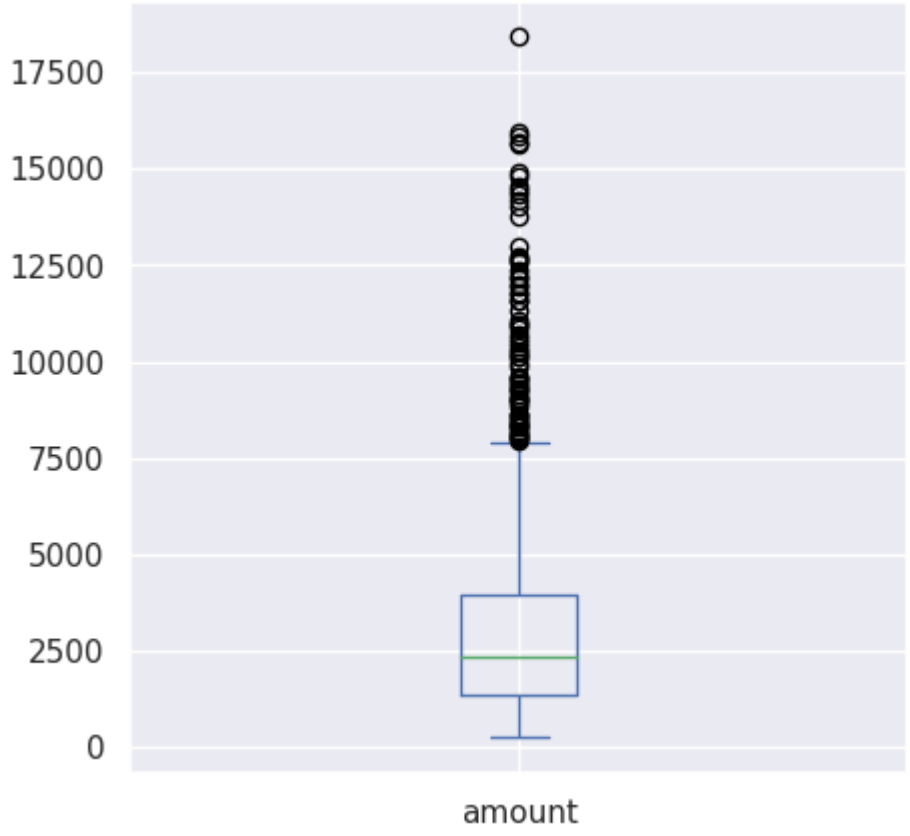


En el pánel anterior observamos que la transformación logaritmica mejora la simetría de las distribuciones.

En el siguiente pánel comparamos valores extremos ("outliers") usando un diagrama box-plot, antes y después de la transformación logaritmica para confirmar la mejora.

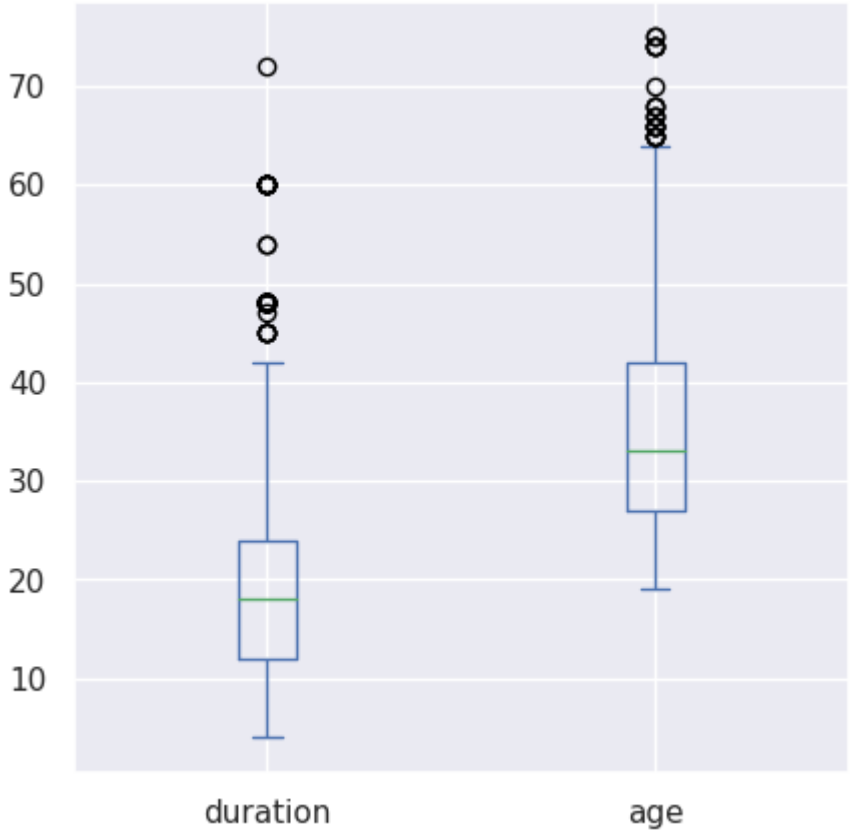
In [ ]:

```
#Redimensiona el tamaño para las siguientes figuras
sns.set(rc={"figure.figsize": (5, 5)})

# Box-plot de los datos numéricos sin transformación
datos["amount"].plot(kind="box", layout=(1,1), figsize=(5,5),sharex=False,sharey=Fal
plt.show()
```

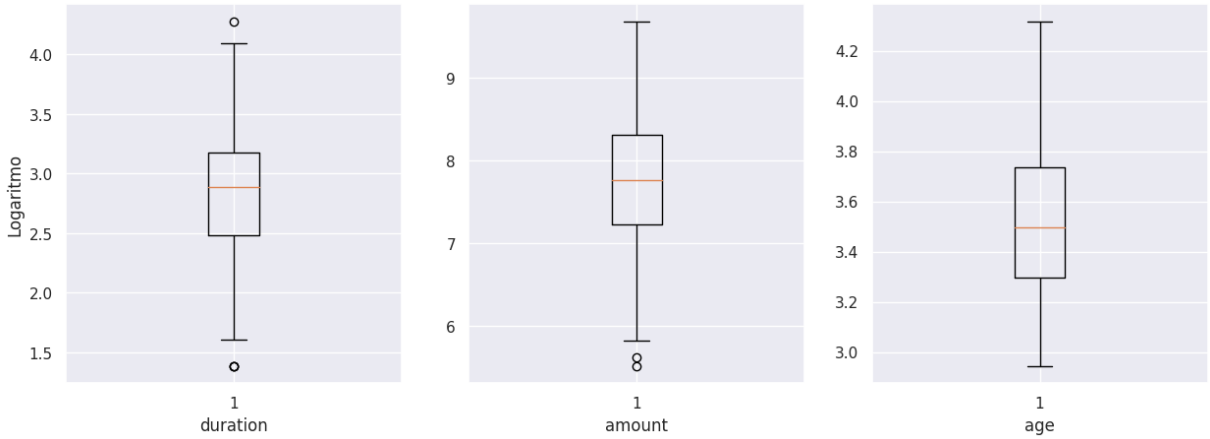


In [ ]:

```
datos[["duration", "age"]].plot(kind="box", layout=(1,1), figsize=(5,5),sharex=False
plt.show()
```



In [ ]:
```
# Box-plot de los datos numéricos usando la transformación logarítmica

#Redimensiona el tamaño para las siguientes figuras
sns.set(rc={"figure.figsize": (15, 5)})

for k in range(0, 3):
    plt.subplot(1, 3, k + 1)
    # Datos originales ----------------------------------------------------------
    Transf0 = (
        X_train[datos_num].iloc[:, k].to_numpy().reshape(-1, 1)
    )  # Se asigna a Transf0 los valores de cada variable "k" sin transformar.
    Transf2 = np.log(
        Transf0
    )  # Se aplica el logaritmo natural a los valores de cada variable "k" sin trans
    plt.boxplot(Transf2)
    plt.xlabel(datos_num[k])
    if k == 0:
        plt.ylabel("Logaritmo")

plt.show()
```



In [ ]:
```
datos.isnull().sum()
```

Out[ ]:
```
status                    0
duration                  0
credit_history            0
purpose                   0
amount                    0
savings                   0
employment_duration       0
installment_rate          0
personal_status_sex       0
other_debtors             0
present_residence         0
property                  0
age                       0
other_installment_plans   0
housing                   0
number_credits            0
job                       0
people_liable             0
telephone                 0
foreign_worker            0
```

```
credit_risk               0
dtype: int64
```

In [ ]:

```python
# En esta sección de código definimos las transformaciones
# Transformaciones a factores categóricos y binarios de entrada:

cat_pipeline = Pipeline(
    steps=[
        ("impModa", SimpleImputer(strategy="most_frequent")),
        ("OneHotE", OneHotEncoder(handle_unknown="ignore")),
    ]
)
cat_pipeline_nombres = [
    "status",
    "credit_history",
    "purpose",
    "savings",
    "personal_status_sex",
    "other_debtors",
    "other_installment_plans",
    "housing",
]


# Transformaciones a factores numéricos de entrada:
num_pipeline = Pipeline(
    steps=[
        ("impMediana", SimpleImputer(strategy="median")),
        ("log", FunctionTransformer(np.log)),
        ("escalaNum", MinMaxScaler()),
    ]
)
num_pipeline_nombres = ["duration", "amount", "age"]


# Transformaciones a factores ordinales de entrada:
ord_pipeline = Pipeline(
    steps=[
        ("impModa", SimpleImputer(strategy="most_frequent")),
    ]
)
ord_pipeline_nombres = [
    "employment_duration",
    "installment_rate",
    "present_residence",
    "property",
    "number_credits",
    "job",
]

bin_pipeline_nombres = ["people_liable", "telephone", "foreign_worker"]


# Conjuntamos las transformaciones que se aplicarán a los datos de entrada:
columnasTransformer = ColumnTransformer(
    transformers=[
        ("catpipe", cat_pipeline, cat_pipeline_nombres+bin_pipeline_nombres),
        ("numpipe", num_pipeline, num_pipeline_nombres),
        ("ordpipe", ord_pipeline, ord_pipeline_nombres),
```

```
    ],
    remainder="passthrough",
)
```

Ejercicio-4.

Entrenamiento usando validación cruzada

Modelos:

- Regresión Logística,
- Árbol de Decisión, y
- Bosque Aleatorio.

Llevarás un entrenamiento usando validación cruzada entre los siguientes tres modelos de aprendizaje automático: Regresión Logística, Árbol de Decisión y Bosque Aleatorio. Deberás llevar a cabo el entrenamiento de los tres de manera conjunta usando un ciclo FOR. Recuerda aplicar las transformaciones que definiste en tu Pipeline. El entrenamiento debe ser con las siguientes características:

a. Usa los parámetros predeterminados de cada modelo.

In [ ]:

```
def get_models():
    modelos = list()
    nombres = list()

    # LR - Regresión Logística:
    modelos.append(LogisticRegression(max_iter=3000))
    nombres.append("LR")

    # DT - Árbol de Decisión:
    modelos.append(DecisionTreeClassifier())
    nombres.append("DT")

    # RF - Random Forest:
    modelos.append(RandomForestClassifier())
    nombres.append("RF")

    return modelos, nombres
```

4.b. En cada iteración deben calcularse todas las siguientes métricas: accuracy, precision, recall, f1-score y Gmean. Todas estas métricas deben ser funciones que tú mismo debes definir (Es decir, no usar las funciones de dichas métricas que te proporciona scikit-learn. Sin embargo, sí puedes usar la

información regresada por el método confusion_matrix() de scikit-learn para definir las métricas).

In [ ]:

```
# Funciones para generar scores
def accuracy(yreal, ypred):
    cm = confusion_matrix(yreal, ypred)

    tot = cm.sum()

    vp = cm[1, 1]
    vn = cm[0, 0]

    score = (vp + vn) / tot

    return score


def precision(yreal, ypred):
    cm = confusion_matrix(yreal, ypred)

    tot = cm.sum()

    fp = cm[0, 1]
    vp = cm[1, 1]

    score = vp / (vp + fp)

    return score


def recall(yreal, ypred):
    cm = confusion_matrix(yreal, ypred)

    tot = cm.sum()

    fn = cm[1, 0]
    vp = cm[1, 1]

    score = vp / (vp + fn)

    return score


def f1(yreal, ypred):
    cm = confusion_matrix(yreal, ypred)

    tot = cm.sum()

    fp = cm[0, 1]
    fn = cm[1, 0]
    vp = cm[1, 1]

    score = 2 * vp / (2 * vp + fp + fn)

    return score


def gmean(yreal, ypred):
```

```
    cm = confusion_matrix(yreal, ypred)

    tot = cm.sum()

    vn = cm[0, 0]
    fp = cm[0, 1]
    fn = cm[1, 0]
    vp = cm[1, 1]

    recall = vp / (vp + fn)
    specifity = vn / (vn + fp)

    score = np.sqrt(recall * specifity)

    return score
```

In [ ]:

```
# Código de prueba de muestras funciones de métricas
#y_true = ['cat', 'dog', 'cat', 'cat', 'dog', 'dog','dog','cat','cat','dog']
#y_pred = ['dog', 'dog', 'cat', 'cat','dog', 'cat','dog','dog','cat','dog']
y_true = [2, 0, 2, 2, 0, 2]
y_pred = [0, 0, 2, 2, 0, 2]
cm = confusion_matrix(y_true, y_pred)
print(cm)
print('accuracy',accuracy(y_true, y_pred))
print('precision',precision(y_true, y_pred))
print('recall',recall(y_true, y_pred))
print('f1',f1(y_true, y_pred))
print('gmean',gmean(y_true, y_pred))
```

```
[[2 0]
 [1 3]]
accuracy 0.8333333333333334
precision 1.0
recall 0.75
f1 0.8571428571428571
gmean 0.8660254037844386
```

4.c. Usar validación cruzada estratificada con 5 particiones y con 3 repeticiones.

4.d. Imprimir el valor de todas estas métricas, tanto para los datos de entrenamiento, como para los de validación. Así como los diagramas de caja y bigotes de los tres modelos con la métrica "recall". ¿Alguno de los modelos está subentrenado o sobreentrenado? Justifica tu respuesta.

Después de la evaluación podemos concluir que los modelos: árbol de decisión y bosque aleatorio se encuentran sobre entrenados. Esto lo sabemos al comparar las métricas de entrenamiento y validación en donde existe una varianza alta y un sesgo muy pequeño.

Para el modelo de regresión logística identificamos un desempeño con ligero sub entrenamiento ya que el f1 score de entrenamiento tiene una tendencia negativa.

In [ ]:

```
modelos, nombres = get_models()  # cargamos los modelos a comparar
resultados = list()

#Define la validación cruzada con los paràmetros requeridos:
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=45)

for i in range(len(modelos)):

    pipeline = Pipeline(steps=[("ct", columnasTransformer), ("m", modelos[i])])
    metricas = {
        "accuracy": make_scorer(accuracy),
        "precision": make_scorer(precision),
        "recall": make_scorer(recall),
        "f1": make_scorer(f1),
        "gmean": make_scorer(gmean),
    }

    scores = cross_validate(
        pipeline, X_train, Y_train, scoring=metricas, cv=cv, return_train_score=True
    )

    resultados.append(scores)
    print('==============================\n',nombres[i],':\n=======================
    print('ENTRENAMIENTO:')
    print('mean Accuracy: %.3f (%.4f)\nmean Precision: %.3f (%.4f)\nmean Recall: %.3
                                                                    np.mean(sc
                                                                    np.std(scc
                                                                    np.mean(sc
                                                                    np.std(scc
                                                                    np.mean(sc
                                                                    np.std(scc
                                                                    np.mean(sc
                                                                    np.std(scc
                                                                    np.mean(sc
                                                                    np.std(scc
                                                                    ))

    print('VALIDACION (interna al método de validación cruzada):')
    print('mean Accuracy: %.3f (%.4f)\nmean Precision: %.3f (%.4f)\nmean Recall: %.3
                                                                    np.mean(sc
                                                                    np.std(scc
                                                                    np.mean(sc
                                                                    np.std(scc
                                                                    np.mean(sc
                                                                    np.std(scc
                                                                    np.mean(sc
                                                                    np.std(scc
                                                                    np.mean(sc
                                                                    np.std(scc
                                                                    np.mean(sc
                                                                    np.std(scc
                                                                    ))
```

```
==============================
 LR :
==============================
ENTRENAMIENTO:
mean Accuracy: 0.782 (0.0095)
mean Precision: 0.810 (0.0083)
mean Recall: 0.899 (0.0069)
```

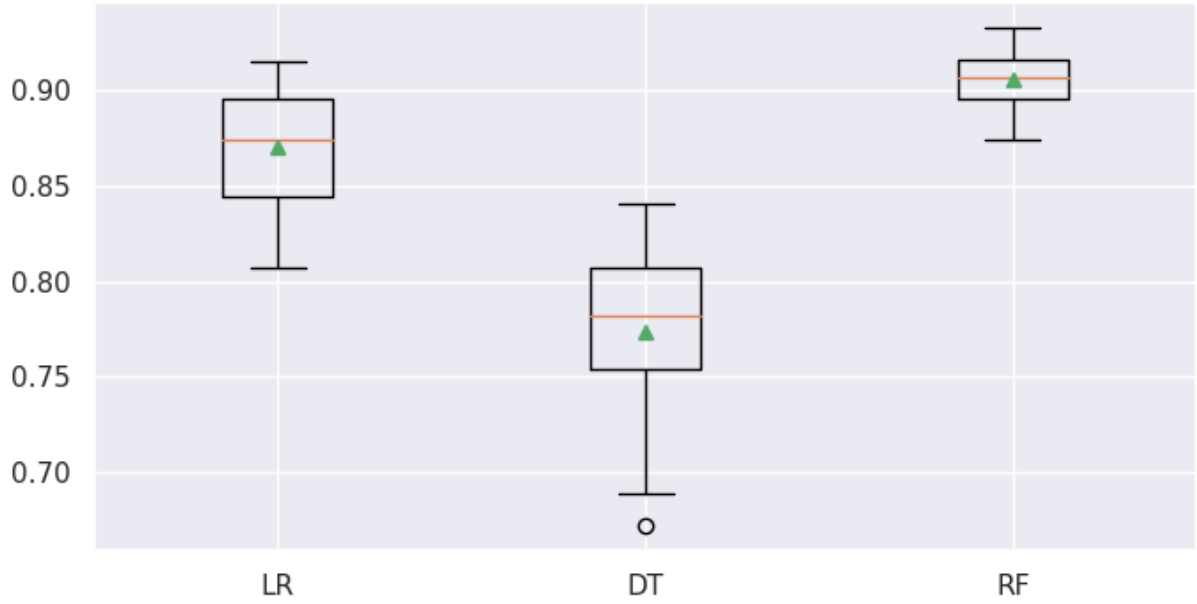
```
mean F1-score: 0.852 (0.0060)
Gmean: 0.678 (0.0183)

VALIDACION (interna al método de validación cruzada):
mean Accuracy: 0.747 (0.0258)
mean Precision: 0.789 (0.0174)
mean Recall: 0.870 (0.0309)
mean F1-score: 0.827 (0.0186)
Gmean: 0.632 (0.0396)

==============================
 DT :
==============================
ENTRENAMIENTO:
mean Accuracy: 1.000 (0.0000)
mean Precision: 1.000 (0.0000)
mean Recall: 1.000 (0.0000)
mean F1-score: 1.000 (0.0000)
Gmean: 1.000 (0.0000)

VALIDACION (interna al método de validación cruzada):
mean Accuracy: 0.682 (0.0345)
mean Precision: 0.772 (0.0266)
mean Recall: 0.773 (0.0466)
mean F1-score: 0.772 (0.0280)
Gmean: 0.598 (0.0530)

==============================
 RF :
==============================
ENTRENAMIENTO:
mean Accuracy: 1.000 (0.0000)
mean Precision: 1.000 (0.0000)
mean Recall: 1.000 (0.0000)
mean F1-score: 1.000 (0.0000)
Gmean: 1.000 (0.0000)

VALIDACION (interna al método de validación cruzada):
mean Accuracy: 0.748 (0.0240)
mean Precision: 0.773 (0.0198)
mean Recall: 0.906 (0.0169)
mean F1-score: 0.834 (0.0143)
Gmean: 0.586 (0.0579)
```

diagramas de caja y bigotes de los tres modelos con la métrica "recall".

In [ ]:

```python
#Imprimimos las metricas
sns.set(rc={'figure.figsize':(8,4)})

bprecall = list()

for i in range (len(resultados)):
  rr = resultados[i]['test_recall']
  bprecall.append(rr)

plt.boxplot(bprecall, labels=nombres, showmeans=True)
```

```
plt.show()
```



¿Alguno de los modelos está subentrenado o sobreentrenado? Justifica tu respuesta.

FALTA

En particular obtengamos algunas de las llamadas curvas de aprendizaje para algunos de estos casos. En dada gráfico debes incluir tus comentarios sobre el modelo generado:

i. Obtener las curvas de aprendizaje (learning_curve) en la cual se va incrementando el tamaño de la muestra para el modelo de regresión Logística con sus hiperparámetros predeterminados. Utilizar al menos 20 puntos en la partición de los conjuntos de entrenamiento y la métrica "f1-score", como evaluación del desempeño de dicha función "learning_curve()".

iii. Obtener las curvas de aprendizaje (learning_curve) en la cual se va incrementando el tamaño de la muestra para el modelo de regresión bosque aleatorio (random forest) con su hiperparámetros predeterminados. Utilizar al menos 20 puntos en la partición de los conjuntos de entrenamiento y la métrica "recall", como evaluación del desempeño del modelo.

In [ ]:

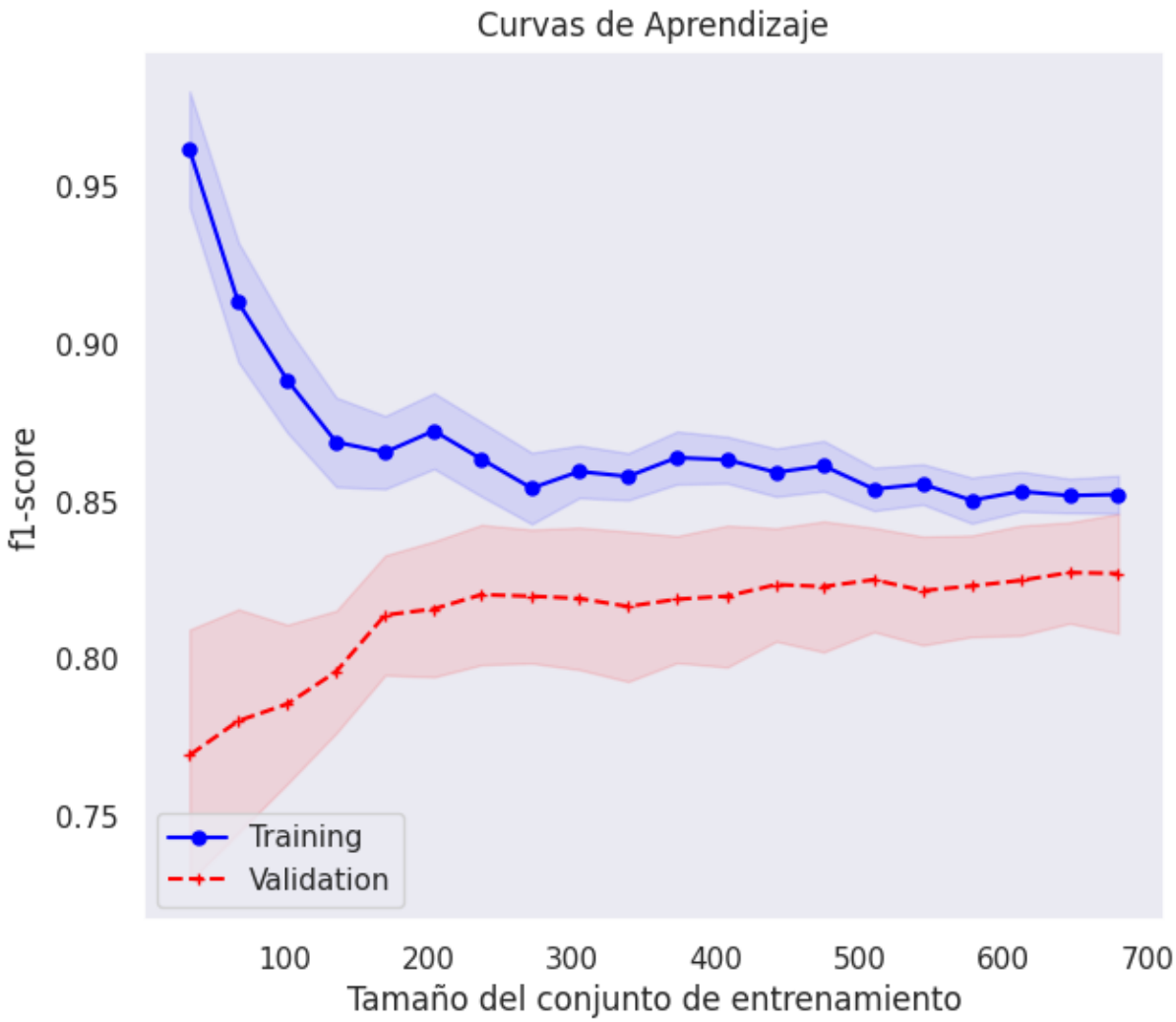
```
def mi_LearningCurvePlot(train_sizes, train_scores, val_scores, eje_y):

    # Argumentos de entrada de la función mi_LearningCurvePlot:
    #      train_sizes : número de observaciones en el conjunto de entrenamiento.
    #      train_scores : Exactitud de cada partición en el proceso de Validación-Cr
```

```
    #                     La dimensión de este conjunto es (pxq)
    #                                     donde p="número de particiones de manera i
    #                                           q="número de particiones de VC" * "n
    #      val_scores : Exactitud de cada partición en el proceso de Validación-Cruz
    #                   Es de la misma dimensión que los train_scores.
    # Output: la salida es el gráfico con las curvas de aprendizaje.

    # Obtenemos los promedios y desviaciones estándar de cada renglón de los resulta
    # La dimensión de cada uno es p="número de particiones de manera incremental del
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
    val_mean = np.mean(val_scores, axis=1)
    val_std = np.std(val_scores, axis=1)

    # Graficamos las curvas de aprendizaje incluyendo una región indicando la desvia
    plt.figure(figsize=(7,6))
    plt.plot(train_sizes, train_mean, color='blue', marker='o', markersize=5, label=
    plt.fill_between(train_sizes, train_mean + train_std, train_mean - train_std, al

    plt.plot(train_sizes, val_mean, color='red', marker='+', markersize=5, linestyle
    plt.fill_between(train_sizes, val_mean + val_std, val_mean - val_std, alpha=0.1,

    plt.title('Curvas de Aprendizaje')
    plt.xlabel('Tamaño del conjunto de entrenamiento')
    plt.ylabel(eje_y)
    plt.grid()
    plt.legend(loc='lower left')
    plt.show()
```

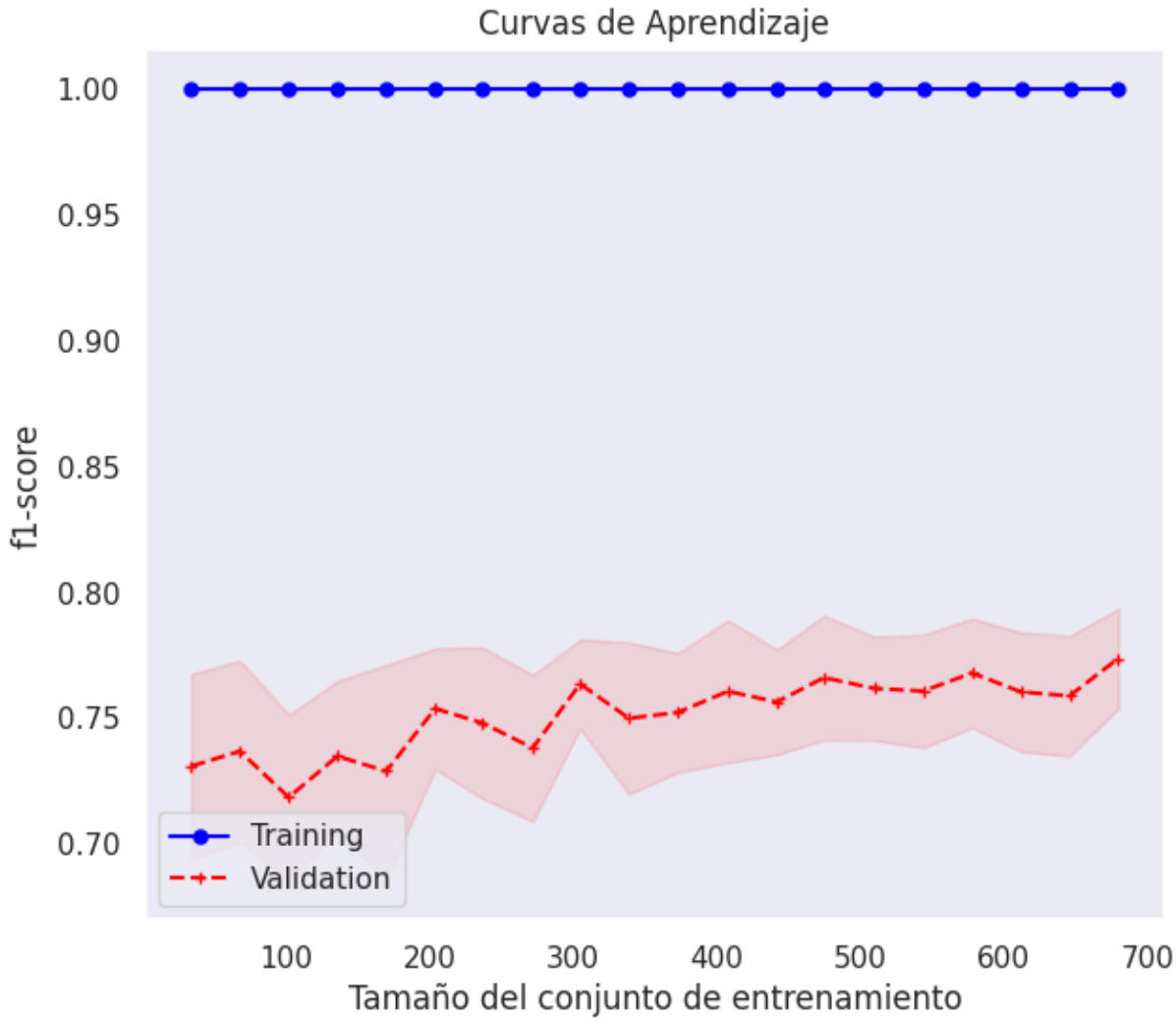
In [ ]:

```
delta_train_sz = np.linspace(.05, 1, 20)

for i in range(len(modelos)):
    #CODIGO DE PRUEBA
    #pipeline = Pipeline(steps=[('escalar',StandardScaler()),('modelo',modelos[i]),]
    metricas = {"f1": make_scorer(f1)}

    pipeline = Pipeline(steps=[("ct", columnasTransformer), ("m", modelos[i])])
#    metricas = {
#        "accuracy": make_scorer(accuracy),
#        "precision": make_scorer(precision),
#        "recall": make_scorer(recall),
#        "f1": make_scorer(f1),
#        "gmean": make_scorer(gmean),
#    }

    tr_sizes, tr_scores, val_scores = learning_curve(estimator = pipeline,
                                                     X = X_train,
                                                     y = Y_train,
                                                     scoring=make_scorer(f1),
                                                     cv = cv,
                                                     train_sizes = delta_train_sz
                                                     random_state=45, n_jobs=-1)

    # Obtenemos el gráfico con las curvas de aprendizaje:
    print('--------------------------\n','MODELO: '+nombres[i],'\n----------------
    mi_LearningCurvePlot(tr_sizes, tr_scores, val_scores, 'f1-score')
```

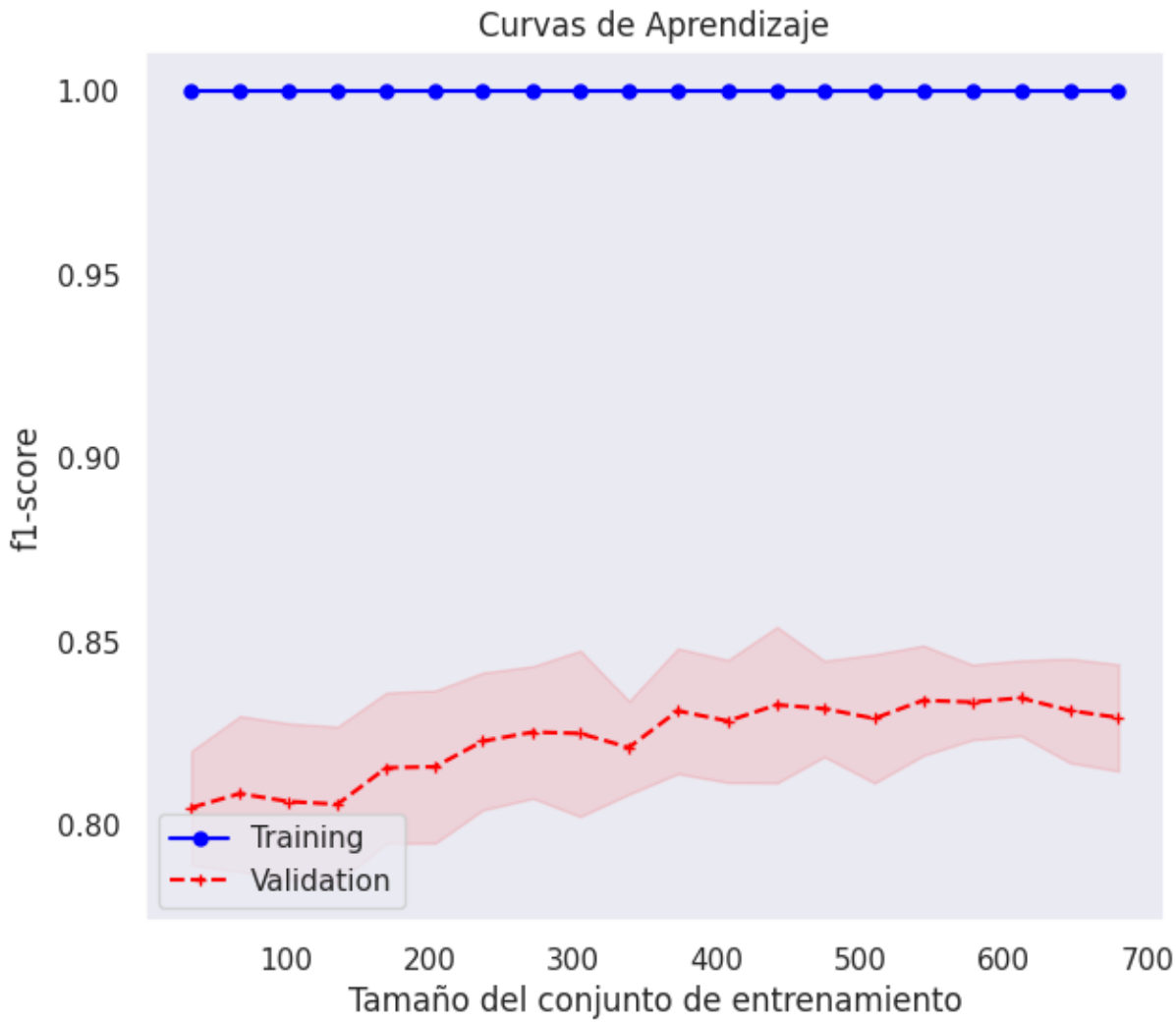
MODELO: LR



MODELO: DT


Curvas de Aprendizaje
1.00
0.95
0.90
0.85
0.80
0.75
0.70
f1-score
Training
Validation
100
200
300
400
500
600
700
Tamaño del conjunto de entrenamiento

```
----------------------------
 MODELO: RF
----------------------------
```

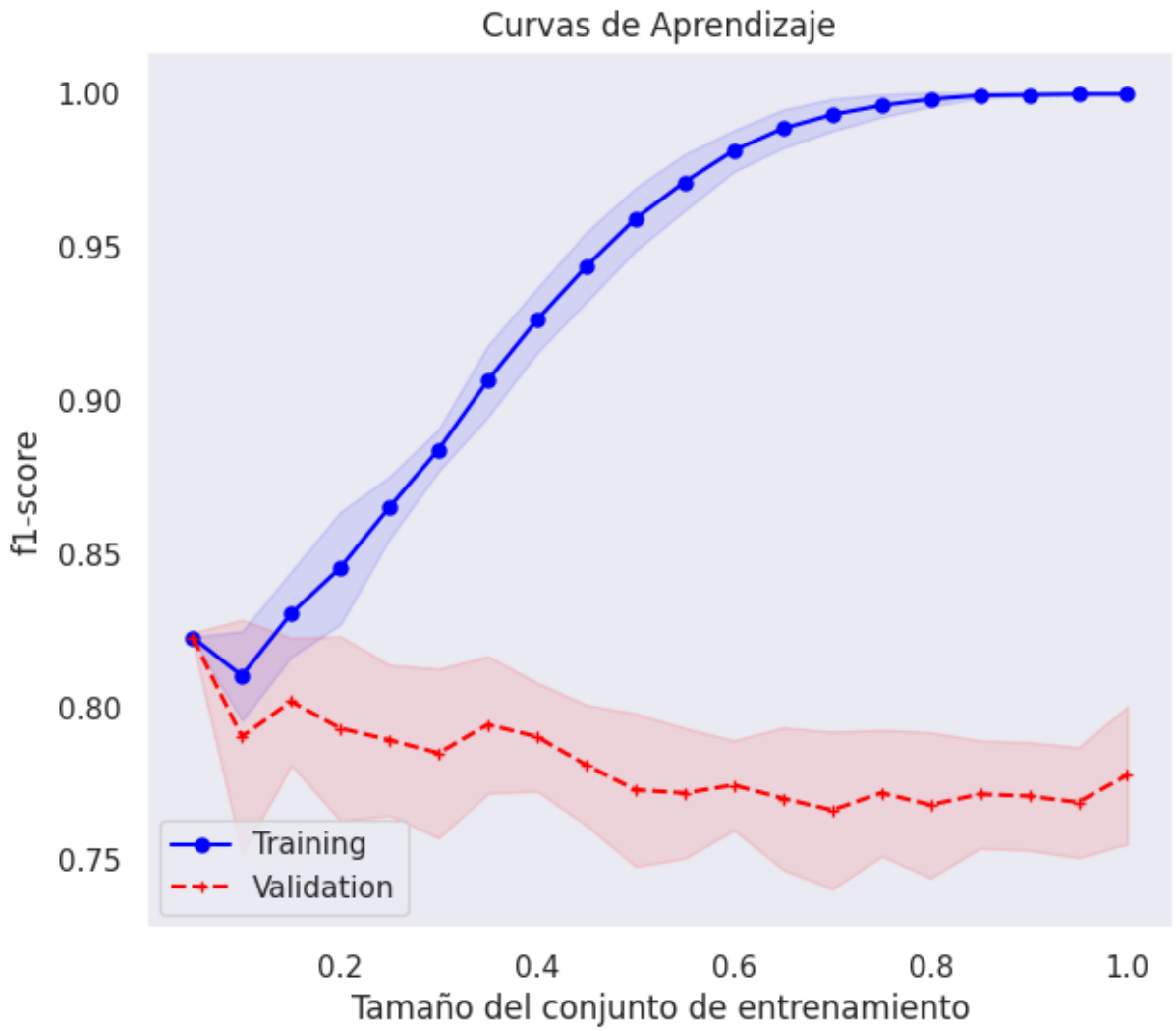


ii. Obtener las curvas de validación (validation_curve) en la cual se va incrementando la complejidad del hiperparámetro "max_depth" para el modelo de árbol de decisión con sus hiperparámetros predeterminados. Utilizar valores de máxima profundidad desde 1 hasta 20 y con la métrica "f1-score" para la evaluación del desempeño del modelo.

In [ ]:

```
modelo = modelos[1]
pipeline = Pipeline(steps=[("ct", columnasTransformer), ("m", modelo)])

max_depth_range = range(1, 21)

#Usado para depurar código cuando usamos validation_curve
#print(pipeline.get_params().keys())

train_scores, test_scores = validation_curve(
    estimator=pipeline,
    X=X_train,
    y=Y_train,
    param_name='m__max_depth',
    param_range=max_depth_range,
    scoring=make_scorer(f1),
    cv=cv, n_jobs=-1
```

```
    )

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.figure(figsize=(7,6))

plt.plot(delta_train_sz, train_mean, color='blue', marker='o', markersize=5, label='
plt.fill_between(delta_train_sz, train_mean + train_std, train_mean - train_std, alp

plt.plot(delta_train_sz, test_mean, color='red', marker='+', markersize=5, linestyle
plt.fill_between(delta_train_sz, test_mean + test_std, test_mean - test_std, alpha=0

plt.title('Curvas de Aprendizaje')
plt.xlabel('Tamaño del conjunto de entrenamiento')
plt.ylabel('f1-score')
plt.grid()
plt.legend(loc='lower left')
plt.show()
```



Ejercicio-5.

Mejoramiento de los hiperparámetros y sobreentrenamiento

a. Para el modelo de regresión logística realizar el entrenamiento buscando sus mejores hiperparámetros con GridSearchCV(). Los hiperparámetros que debes incluir en su búsqueda deben ser al menos los siguientes: C, solver, class_weight y penalty. En este caso deberás usar la métrica (scoring) "f1-score". Imprime la mejor combinación de parámetros obtenidos, así como el valor del mejor desempeño (score) obtenido con la métrica f1. ¿Cuál es la utilidad de la métrica "f1-score"? Incluye tus conclusiones.

NOTA: Toma en cuenta que no todas las combinaciones de "solver" y "penalty" son posibles, para que lo tomes en cuenta al momento de realizar la búsqueda. Revisa la documentación.

In [ ]:

```
mi_modelo_LR = LogisticRegression(max_iter=3000, random_state=45)

#PRUEBAS DE DIFERENTES DICCIONARIOS:
#Lo hicimos de esta manera dado que algunas combinaciones son incompatibles
#Al final escogimos el mejor de ellos e hicimos el GridSearchCV sobre variaciones de

#El diccionario siguiente ya esta optimizado para el parametro C y solvers
#dicc_grid = {'C':[0.01, 0.08, 0.09, 0.1, 0.11, 0.12, 0.4, 1.0, 10], 'solver':['newt
#Ahora probamos valores de penalty.
#dicc_grid = {'C':[0.11, 0.12], 'solver':['newton-cg','lbfgs', 'sag','saga'], 'penal
#dicc_grid = {'C':[0.11, 0.12], 'solver':['liblinear'], 'penalty':['l1','l2']}
#dicc_grid = {'C':[0.11, 0.12], 'solver':['saga'], 'penalty':['l1','l2']}
#dicc_grid = {'C':[0.11, 0.12], 'solver':['saga'], 'penalty':['elasticnet'],'l1_rati

dicc_grid = {'C':[0.09, 0.1, 0.11, 0.12], 'solver':['saga'], 'penalty':['elasticnet'

grid = GridSearchCV(estimator=mi_modelo_LR, param_grid=dicc_grid, cv=cv, scoring=mak
# Transformamos los datos de entrada:
Xx = columnasTransformer.fit_transform(X_train)
# Llevamos a cabo el proceso de etrenamiento con validación-cruzada y búsqueda de ma
# Observa que de acuerdo a las opciones incluidas en la malla, se estarán realizando
# combinaciones diferentes, además de las (10)(5)=50 particiones de la validación-cr
# lo cual implica también un mayor tiempo de entrenamiento.
grid.fit(Xx, np.ravel(Y_train))
print('f1-score a superar:\n------------------')
print('mean f1-score: %.3f (%.4f)\n' % (np.mean(scores['test_f1']), np.std(scores['t
print('Mejor valor de f1-score obtenido con la mejor combinación:', grid.best_score_
print('Mejor combinación de valores encontrados de los hiperparámetros:', grid.best_
print('Métrica utilizada:', grid.scoring)

#Guardamos los mejores hiperparámetros extraídos del barrido con GridSearchCV
mejores_params_LR = grid.best_params_
```

```
f1-score a superar:
------------------
mean f1-score: 0.834 (0.0143)
```

```
Mejor valor de f1-score obtenido con la mejor combinación: 0.8371845992183143
Mejor combinación de valores encontrados de los hiperparámetros: {'C': 0.11, 'class_
weight': None, 'l1_ratio': 0.25, 'penalty': 'elasticnet', 'solver': 'saga'}
Métrica utilizada: make_scorer(f1)
```

¿Cuál es la utilidad de la métrica “f1-score”?

El f1-score es una métrica de error que mide el desempeño de un modelo calculando la media armónica de la precisión y el recall para la clase positiva minoritaria. Provee resultados precisos para problemas de clasificación balanceados y desbalanceados.

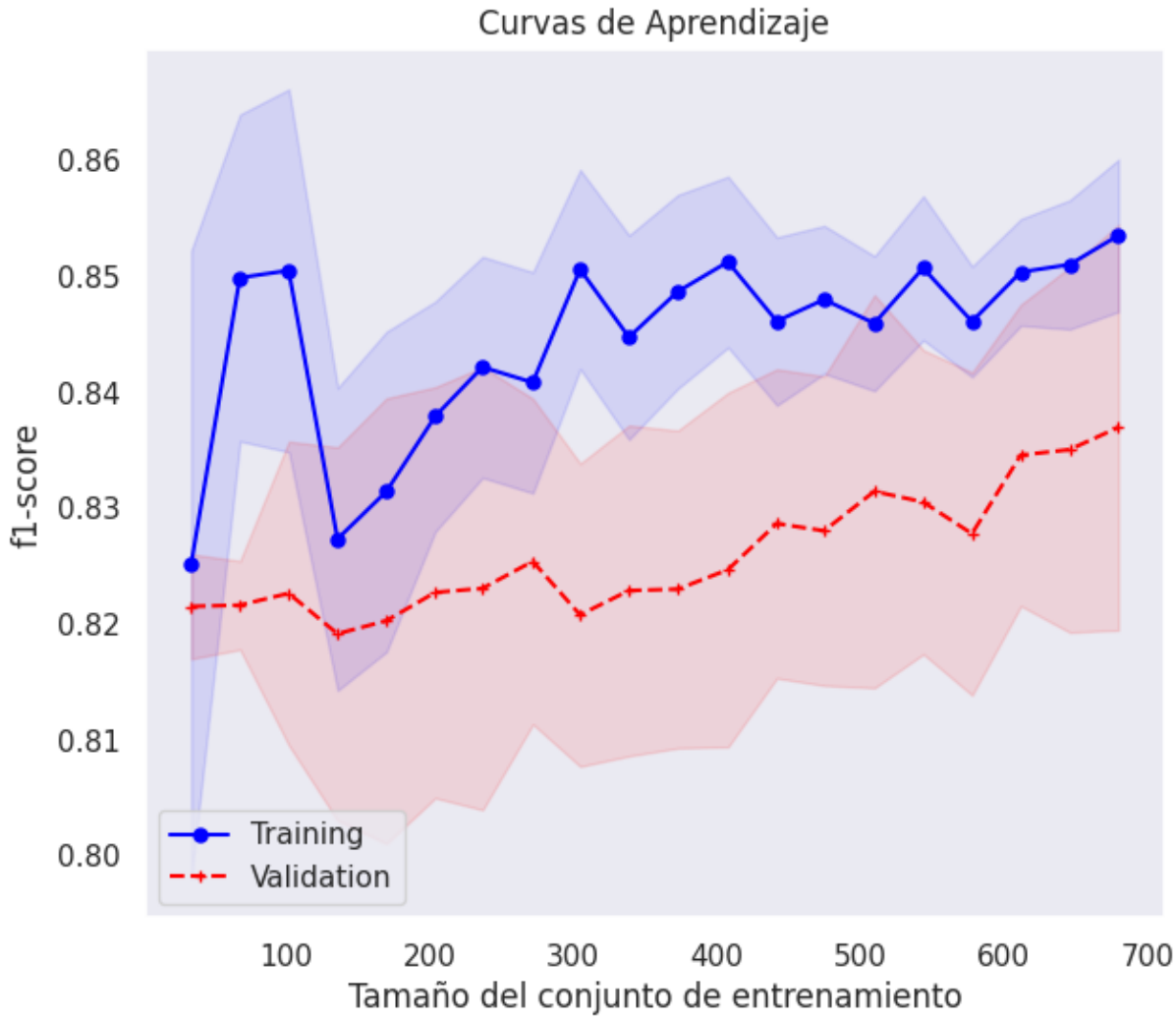
F1 score puede ser interpretado como la habilidad de un modelo para capturar casos positivos y ser preciso en su clasificación. Es una métrica popular para problemas de clasificación binaria donde hay un desequilibrio en la distribución de las clases. Por ejemplo, en un problema de detección de spam, donde solo el 1% de los correos electrónicos son spam, la precisión y el recall son buenas métricas para evaluar un modelo. Sin embargo, si el modelo clasifica todos los correos electrónicos como no spam, tendrá una precisión del 99% y un recall del 0%. En este caso, el f1-score es una mejor métrica para evaluar el modelo, ya que es la media armónica de la precisión y el recall, que penaliza los valores bajos de ambas métricas.

b. Con los mejores valores de los hiperparámetros encontrados con la métrica “f1-score” para el modelo de regresión logística, obtener las curvas de aprendizaje(learning curve), incrementando el tamaño del conjunto de entrenamiento al menos 20 veces. Si lo crees adecuado, puedes hacer los ajustes que consideres adecuados para mejorar el resultado y evitar el sobreentrenamiento o el subentrenamiento.

In [ ]:

```
#Define los tamaños de conjunto de entrenamiento a usar
delta_train_sz = np.linspace(.05, 1, 20)
#Define modelo a trabajar en esta sección
mejor_modelo_LR = LogisticRegression(max_iter=3000, random_state=45,
                                     C=mejores_params_LR.get('C'),
                                     solver=mejores_params_LR.get('solver'),
                                     class_weight=mejores_params_LR.get('class_weigh
                                     penalty=mejores_params_LR.get('penalty'),
                                     l1_ratio=mejores_params_LR.get('l1_ratio'),
)

pipeline = Pipeline(steps=[("ct", columnasTransformer), ("m", mejor_modelo_LR )])
tr_sizes, tr_scores, val_scores = learning_curve(estimator = pipeline,
                                                 X = X_train,
                                                 y = Y_train,
                                                 scoring=make_scorer(f1),
                                                 cv = cv,
                                                 train_sizes = delta_train_sz,
                                                 random_state=45, n_jobs=-1)
# Obtenemos el gráfico con las curvas de aprendizaje:
print('MODELO: LR\n---------------------------')
mi_LearningCurvePlot(tr_sizes, tr_scores, val_scores, 'f1-score')
```

```
MODELO: LR
--------------------------
```



FALTA: Subentrenado....

c. Para el modelo de árbol de decisión (decision tree) realizar el entrenamiento buscando sus mejores hiperparámetros con GridSearchCV(). Los hiperparámetros que debes incluir en su búsqueda deben ser al menos los siguientes: ccp_alpha, criterion, max_depth, min_samples_split y class_weight. En este caso deberás usar la métrica (scoring) "precision". Imprime la mejor combinación de parámetros obtenidos, así como el valor del mejor desempeño (score) obtenido con la métrica "precision". ¿Cuál es la utilidad de la métrica "precision"? Incluye tus conclusiones.

In [ ]:

```
mi_modelo_DT = DecisionTreeClassifier()

#Parámetros requeridos: ccp_alpha, criterion, max_depth, min_samples_split y class_w
#Como definimos primero el Bosque Aleatorio y lo parecido de los algoritmos decidimo
#usar un diccionario muy parecido
dicc_grid = {'ccp_alpha':np.linspace(0.0, 0.04, 3),
             'criterion':['gini', 'entropy'],
```

```
            #No cambiar!!! En la tarea se piden mínimo 10 pruebas de max_depth
            'max_depth':range(1, 21, 2),
            'min_samples_split':range(2, 12),
            'class_weight':[None,'balanced']}

grid = GridSearchCV(estimator=mi_modelo_DT, param_grid=dicc_grid, cv=cv, scoring=mak
# Transformamos los datos de entrada:
Xx = columnasTransformer.fit_transform(X_train)
# Llevamos a cabo el proceso de etrenamiento con validación-cruzada y búsqueda de ma
# Observa que de acuerdo a las opciones incluidas en la malla, se estarán realizando
# combinaciones diferentes, además de las (10)(5)=50 particiones de la validación-cr
# lo cual implica también un mayor tiempo de entrenamiento.
grid.fit(Xx, np.ravel(Y_train))
print('Precisión a superar:\n------------------')
print('mean Precisión: %.3f (%.4f)\n' % (np.mean(scores['test_precision']), np.std(s
print('Mejor valor de Precisión obtenido con la mejor combinación:', grid.best_score
print('Mejor combinación de valores encontrados de los hiperparámetros:', grid.best_
print('Métrica utilizada:', grid.scoring)

#Guardamos los mejores hiperparámetros extraídos del barrido con GridSearchCV
mejores_params_DT = grid.best_params_
```

```
Precisión a superar:
------------------
mean Precisión: 0.773 (0.0198)

Mejor valor de Precisión obtenido con la mejor combinación: 0.8766681445310363
Mejor combinación de valores encontrados de los hiperparámetros: {'ccp_alpha': 0.0,
'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 1, 'min_samples_spli
t': 2}
Métrica utilizada: make_scorer(precision)
```

¿Cuál es la utilidad de la métrica “precision”?

La precisión es la proporción de predicciones positivas que son correctas. De una manera más simple, es la capacidad de un modelo para no etiquetar como positivo un ejemplo que es negativo.

De ese modo, nos sirve para determinar qué cantidad de predicciones positivas son correctas del total de predicciones positivas que se hicieron.

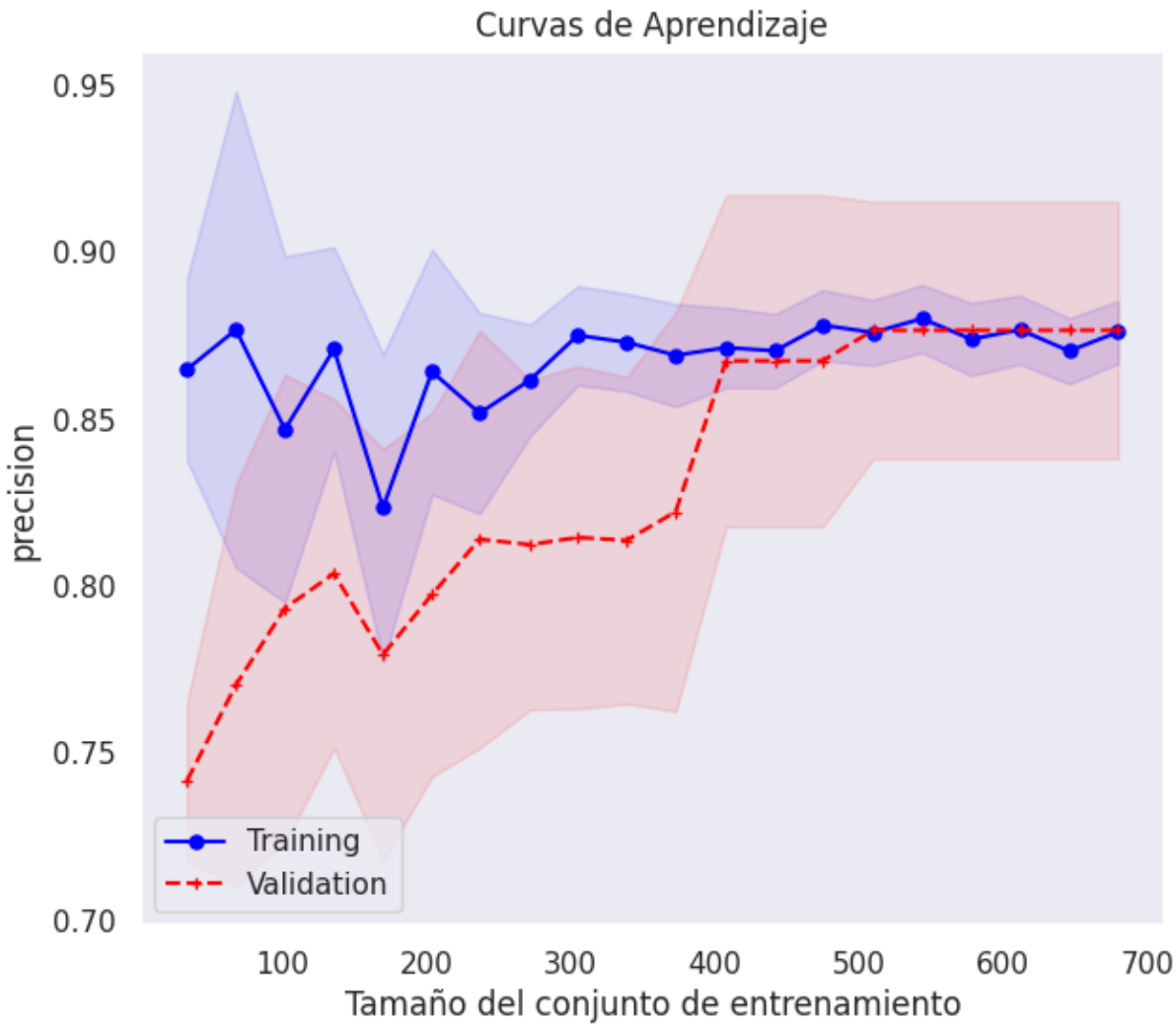
d. Con los mejores valores de los hiperparámetros encontrados con la métrica “precision” para el modelo de árbol de decisión, obtener las curvas de aprendizaje (learning curve), incrementando el tamaño del conjunto de entrenamiento al menos 20 veces. Si lo crees adecuado, puedes hacer los ajustes que consideres adecuados para mejorar el resultado y evitar el sobreentrenamiento o el subentrenamiento.

In [ ]:

```
mejor_modelo_DT = DecisionTreeClassifier(ccp_alpha=mejores_params_DT.get('ccp_alpha'
                                         criterion=mejores_params_DT.get('criterion'),
                                         max_depth=mejores_params_DT.get('max_depth'),
                                         min_samples_split=mejores_params_DT.get('min_samples
                                         class_weight=mejores_params_DT.get('class_weight'))
```

```
delta_train_sz = np.linspace(.05, 1, 20)
pipeline = Pipeline(steps=[("ct", columnasTransformer), ("m", mejor_modelo_DT)])
tr_sizes, tr_scores, val_scores = learning_curve(estimator = pipeline,
                                                 X = X_train,
                                                 y = Y_train,
                                                 #Cambiar a precision
                                                 scoring=make_scorer(precision),
                                                 cv = cv,
                                                 train_sizes = delta_train_sz,
                                                 random_state=45, n_jobs=-1)
# Obtenemos el gráfico con las curvas de aprendizaje:
print('MODELO: DT\n---------------------------')
mi_LearningCurvePlot(tr_sizes, tr_scores, val_scores, 'precision')
```

```
MODELO: DT
---------------------------
```



e. Para el modelo de bosque aleatorio (random forest) realizar el entrenamiento buscando sus mejores hiperparámetros con GridSearchCV(). Los hiperparámetros que debes incluir en su búsqueda deben ser al menos los siguientes: ccp_alpha, criterion, max_depth, min_samples_split y class_weight. En este caso deberás usar la métrica (scoring) "recall". Imprime la mejor

combinación de parámetros obtenidos, así como el valor del mejor desempeño (score) obtenido con la métrica "recall".

NOTA: Toma en cuenta que el método de random forest pude tardar varios minutos en llevar a cabo

In [ ]:

```
mi_modelo_RF = RandomForestClassifier()
#ccp_alpha, criterion, max_depth, min_samples_split y class_weight

#Probamos inicialmente con múltiple diccionarios
#================================================
#dicc_grid = {'ccp_alpha':[0.01,0.02,0.03],'criterion':['gini','entropy']}
#cc_alpha y criterion ya estan optimizadas
#dicc_grid = {'ccp_alpha':[0.01,0.02,0.03],'criterion':['gini','entropy'],'max_depth
#max_depth ya está optimizada
#dicc_grid = {'ccp_alpha':[0.1, 0.2],
#             'criterion':['gini','entropy'],
#             'max_depth':[6, 7, 8, 9, 10],
#             'min_samples_split':[6, 7, 8, 9]}
#se optimiza min_samples_split y se finaliza con class_weight
#dicc_grid = {'ccp_alpha':np.linspace(0.0, 0.0004, 3),
#             'criterion':['gini',],
#             'max_depth':range(1, 21),
#             'min_samples_split':[4],
#             'class_weight':[None, 'balanced_subsample','balanced']}
#Uno de los mejores:
# dicc_grid = {'ccp_alpha':np.linspace(0.0, 0.0004, 2),
#              'criterion':['gini',],
            #  'max_depth':[1],
            #  'min_samples_split':[4],
            #  'class_weight':[None]}
#FINALES
dicc_grid = {'ccp_alpha':[0.0, 0.0001, 0.001, 0.01, 0.01],
             'criterion':['gini','entropy'],
             'max_depth':range(1,21,2),
             'min_samples_split':range(2, 12, 2),
             'class_weight':[None, 'balanced_subsample','balanced']}
#================================================

grid = GridSearchCV(estimator=mi_modelo_RF, param_grid=dicc_grid, cv=cv, scoring=mak
# Transformamos los datos de entrada:
Xx = columnasTransformer.fit_transform(X_train)
# Llevamos a cabo el proceso de entrenamiento con validación-cruzada y búsqueda de m
# Observa que de acuerdo a las opciones incluidas en la malla, se estarán realizando
# combinaciones diferentes, además de las (10)(5)=50 particiones de la validación-cr
# lo cual implica también un mayor tiempo de entrenamiento.
grid.fit(Xx, np.ravel(Y_train))
print('Recall a superar:\n------------------')
print('mean recall: %.3f (%.4f)\n' % (np.mean(scores['test_recall']), np.std(scores[
print('Mejor valor de recall obtenido con la mejor combinación:', grid.best_score_)
print('Mejor combinación de valores encontrados de los hiperparámetros:', grid.best_
print('Métrica utilizada:', grid.scoring)

#Modelo con los mejores hiperparámetros extraídos del del barrido con GridSearchCV
mejor_modelo_RF = RandomForestClassifier(ccp_alpha=grid.best_params_.get('ccp_alpha'
                                         criterion=grid.best_params_.get('criterion'),
                                         max_depth=grid.best_params_.get('max_depth'),
```

```
                                    min_samples_split=grid.best_params_.get('min_samples
                                    class_weight=grid.best_params_.get('class_weight'))

#Guardamos los mejores hiperparámetros extraídos del barrido con GridSearchCV
mejores_params_RF = grid.best_params_
```

```
Recall a superar:
-----------------
mean recall: 0.906 (0.0169)

Mejor valor de recall obtenido con la mejor combinación: 1.0
Mejor combinación de valores encontrados de los hiperparámetros: {'ccp_alpha': 0.0,
'class_weight': None, 'criterion': 'gini', 'max_depth': 1, 'min_samples_split': 2}
Métrica utilizada: make_scorer(recall)
```

¿Cuál es la utilidad de la métrica "recall"?

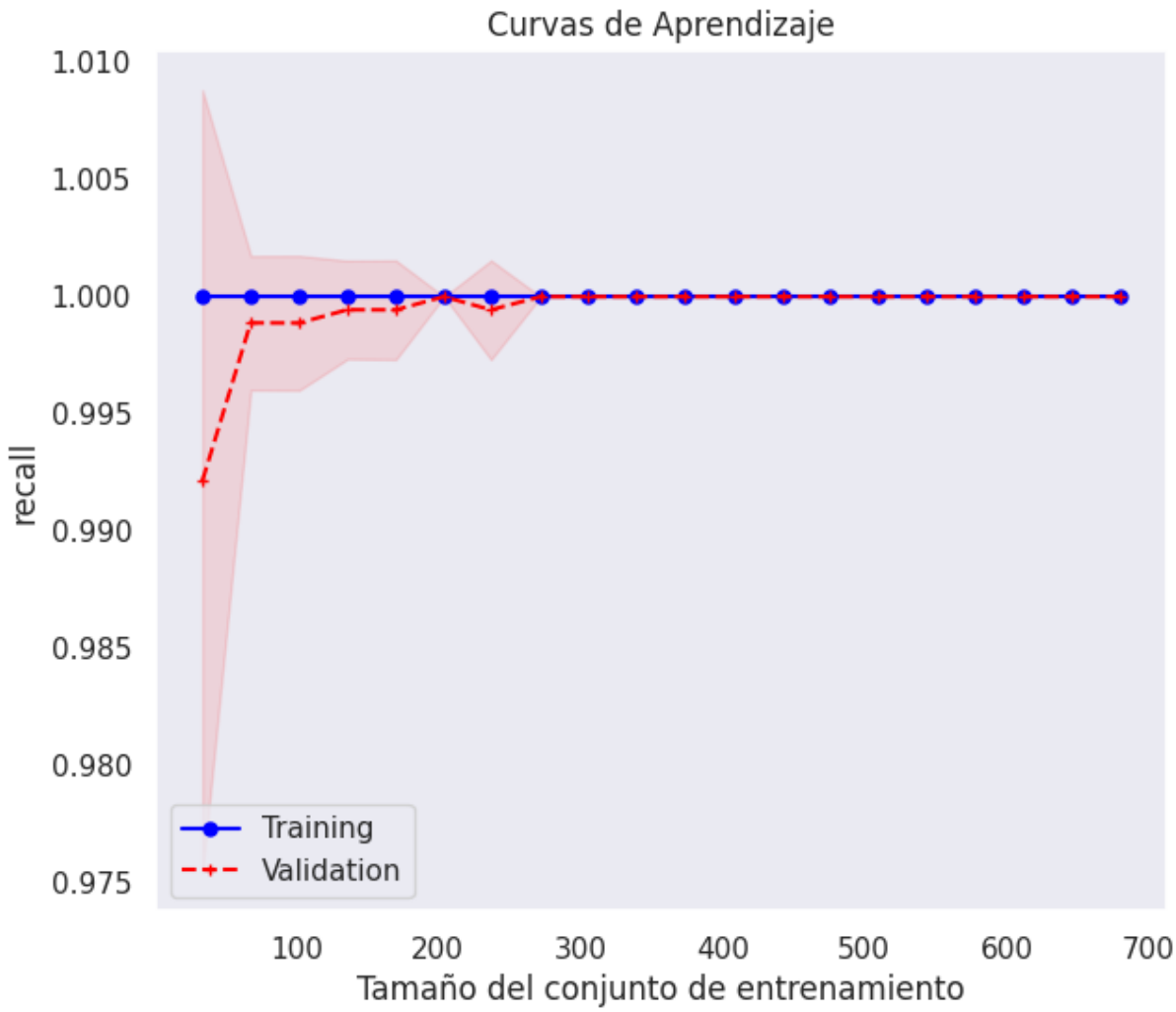
El recall es la proporción de verdaderos positivos que son correctos. De una manera más simple, es la capacidad de un modelo para encontrar todos los ejemplos positivos. A diferencia de la precisión, esta métrica nos dice, del total de los verdaderos positivos reales, qué cantidad de ellos fueron correctamente etiquetados.

f. Con los mejores valores de los hiperparámetros encontrados con la métrica "recall" para el modelo de bosque aleatorio, obtener las curvas de validación (validation curve), incrementando la complejidad del modelo a través del hiperparámetro "max_depth" con al menos 10 valores. Si lo crees adecuado, puedes hacer los ajustes que consideres adecuados para mejorar el resultado y evitar el sobreentrenamiento o el subentrenamiento.

In [ ]:

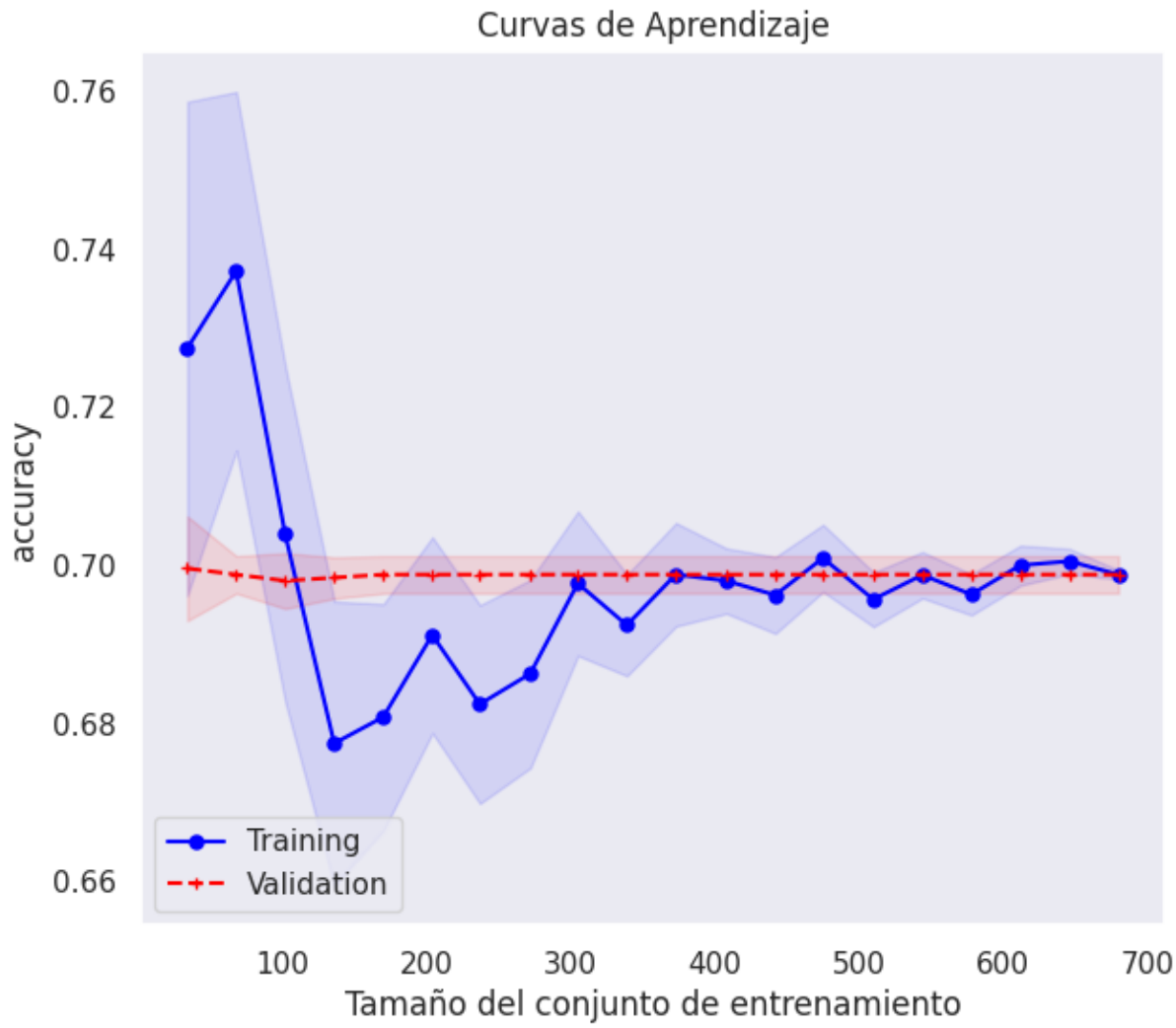
```
delta_train_sz = np.linspace(.05, 1, 20)

pipeline = Pipeline(steps=[("ct", columnasTransformer), ("m", mejor_modelo_RF)])
tr_sizes, tr_scores, val_scores = learning_curve(estimator = pipeline,
                                                 X = X_train,
                                                 y = Y_train,
                                                 scoring=make_scorer(recall),
                                                 cv = cv,
                                                 train_sizes = delta_train_sz,
                                                 random_state=45,n_jobs=-1)
# Obtenemos el gráfico con las curvas de aprendizaje:
print('MEJOR MODELO RF\n--------------------------')
mi_LearningCurvePlot(tr_sizes, tr_scores, val_scores,'recall')
```

```
MEJOR MODELO RF
--------------------------
```



In [ ]:

```
# No es requerido pero sólo por curiosidad miremos la métrica
# accuracy para este modelo
r_sizes, tr_scores, val_scores = learning_curve(estimator = pipeline,
                                                X = X_train,
                                                y = Y_train,
                                                scoring=make_scorer(accuracy),
                                                cv = cv,
                                                train_sizes = delta_train_sz,
                                                random_state=45,n_jobs=-1)

mi_LearningCurvePlot(tr_sizes, tr_scores, val_scores, 'accuracy')
```



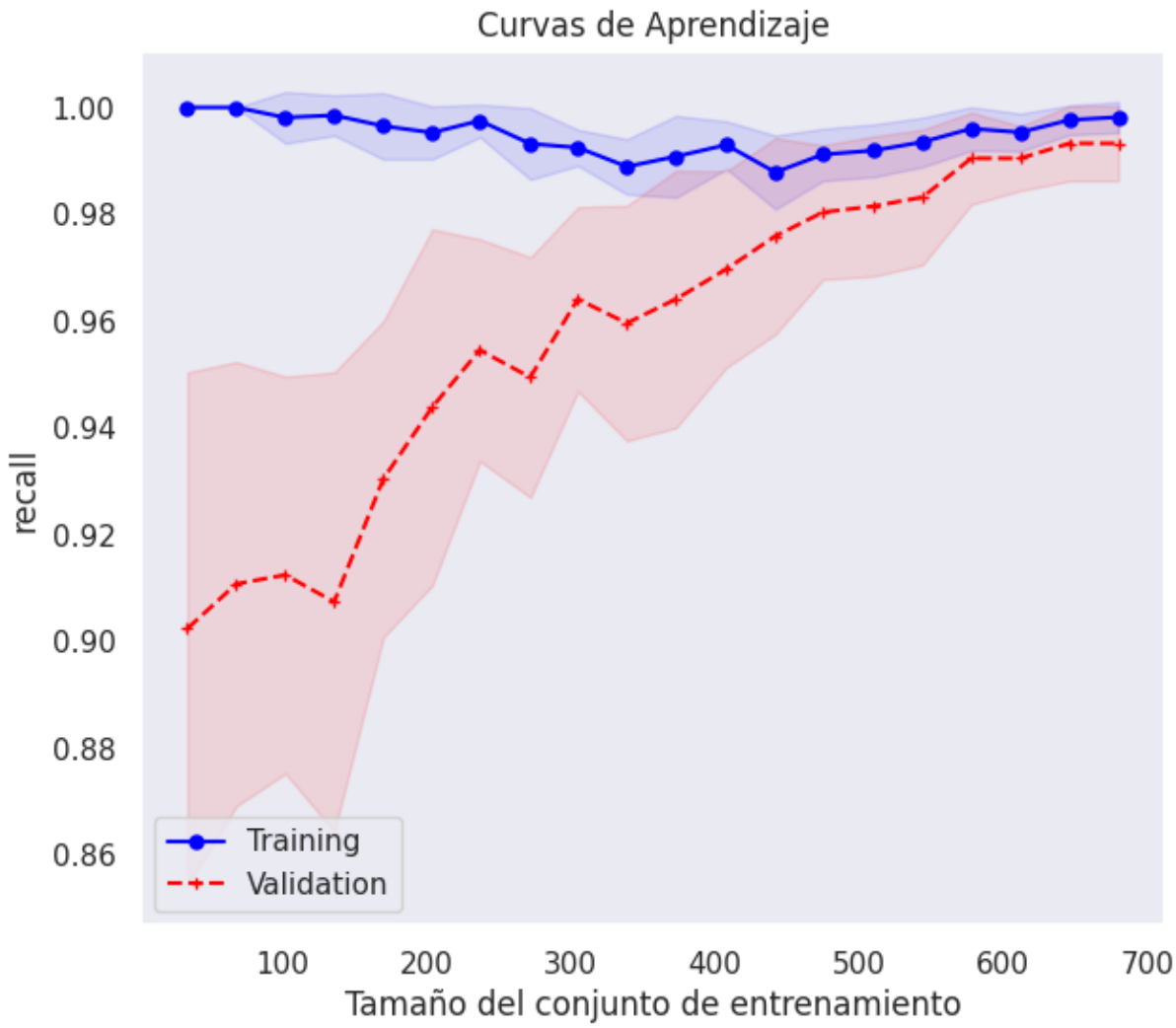
Aquí mostramos un modelo adicional contrastando como al bajar el recall la exactitud sube.

modelo = RandomForestClassifier(ccp_alpha=0.01, criterion='gini', max_depth=7, min_samples_split=5)

In [ ]:

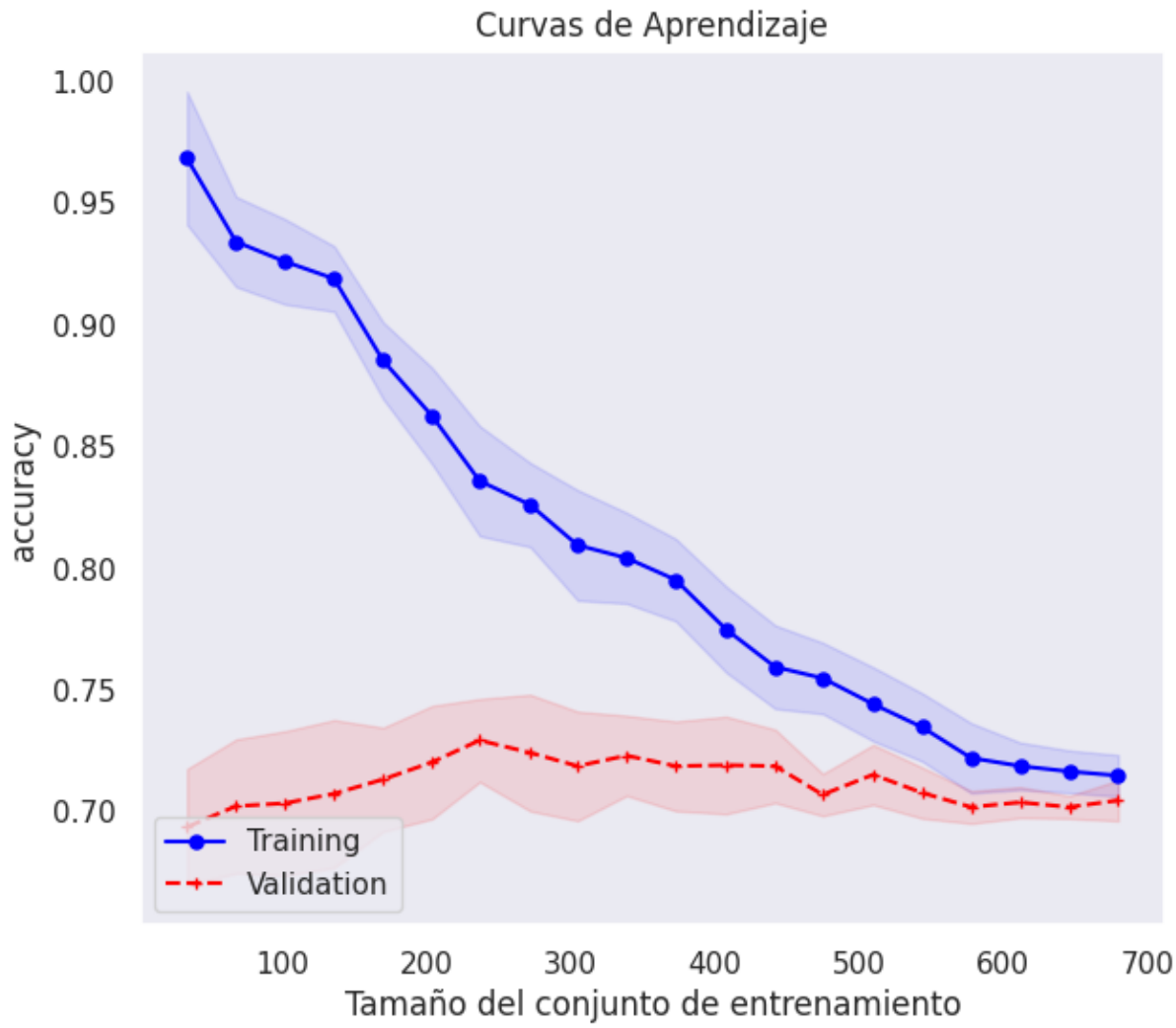
```
modelo_RF_adicional = RandomForestClassifier(ccp_alpha=0.01, criterion='gini', max_d

pipeline = Pipeline(steps=[("ct", columnasTransformer), ("m", modelo_RF_adicional)])
tr_sizes, tr_scores, val_scores = learning_curve(estimator = pipeline,
                                                 X = X_train,
                                                 y = Y_train,
                                                 scoring=make_scorer(recall),
                                                 cv = cv,
                                                 train_sizes = delta_train_sz,
                                                 random_state=40,n_jobs=-1)
# Obtenemos el gráfico con las curvas de aprendizaje:
print('MODELO ADICIONAL RF\n---------------------------')
mi_LearningCurvePlot(tr_sizes, tr_scores, val_scores,'recall')
```

```
MODELO ADICIONAL RF
---------------------------
```



In [ ]:

```
r_sizes, tr_scores, val_scores = learning_curve(estimator = pipeline,
                                                X = X_train,
                                                y = Y_train,
                                                scoring=make_scorer(accuracy),
                                                cv = cv,
                                                train_sizes = delta_train_sz,
                                                random_state=45, n_jobs=-1)

mi_LearningCurvePlot(tr_sizes, tr_scores, val_scores, 'accuracy')
```



¿Cuál es la utilidad de la métrica "recall"? Incluye tus conclusiones.

El recall es la proporción de verdaderos positivos que son correctos. De una manera más simple, es la capacidad de un modelo para encontrar todos los ejemplos positivos. A diferencia de la precisión, esta métrica nos dice, del total de los verdaderos positivos reales, qué cantidad de ellos fueron correctamente etiquetados.

Se observa que aunque el recall es del 100%, la exactitud es sólo del 70%

El recall no es una buena métrica a utilizar con conjuntos de datos desbalanceados....

In [ ]:

Ejercicio-6.

Obtención de los modelos finales

a. Obtener el modelo de regresión logística con los mejores parámetros que hayas encontrado con la métrica f1-score utilizada. Imprimir el valor de dicha métrica e incluye tus conclusiones finales para este caso. Incluir un gráfico del árbol de decisión final obtenido.

In [ ]:

```
#Imprimimos los parámetros del mejor modelo
print('MODELO: REGRESION LOGISTICA\n=======================\nMejores parámetros:\n'
print(mejores_params_LR)

#Entrenamos un nuevo clasificador con el mejor modelo
clf = mejor_modelo_LR.fit(X_train, Y_train)   # Entrenamos con el conjunto de prueba

#Usamos el mismo conjunto de entrenamiento para predecir que se obtendría
predi = clf.predict(X_train)

#Obtenemos el score utilizado para este modelo
print('F1-score en el conjunto de entrenamiento: %.2f' % f1(Y_train, predi))

#Ahora hacemos que el clasificador clasifique nuestro conjunto de pruebas que no ha
#usado hasta el momento.
predi = clf.predict(X_test)

#Finalmente obtenemos nuestro score con el conjunto de pruebas
print('F1-score en el conjunto de prueba: %.2f' % f1(Y_test, predi))
```

```
MODELO: REGRESION LOGISTICA
=======================
Mejores parámetros:

{'C': 0.11, 'class_weight': None, 'l1_ratio': 0.25, 'penalty': 'elasticnet', 'solve
r': 'saga'}
F1-score en el conjunto de entrenamiento: 0.83
F1-score en el conjunto de prueba: 0.83
```

Conclusión Regresión logística

Ambas gráficas (Modelo por defecto VS modelo con parámetros optimizados) no están sub entrenadas, ya que el error no es grande. En ambos casos conforme aumentamos los datos, el score aumenta de igual forma. Podemos concluir que existe un ligero sobre entrenamiento ya que se mantiene una varianza constante entre las gráficas entrenamiento y validación.

También podemos observar que encontramos que 'penalty' = 'elasticnet', eso nos quiere decir que contamos con regularización.

Después de realizar grid search y evaluando exclusivamente (f1-score) pudimos lograr un mejor desempeño del modelo utilizando {'C': 0.11, 'class_weight': None, 'l1_ratio': 0.25, 'penalty': 'elasticnet', 'solver': 'saga'} pasando del mean f1-score: 0.828 a 0.837 un mejoramiento del 0.9% .

Se realizaron dos modelos Sin parámetros 0.828 Después de search grid 0.837

Tomando el segundo modelo mejorado, se realizaron predicciones sobre los datos de prueba y se obtuvo un f1 Score del 0.83 que es un valor aceptable y además indica que el modelo no está sobreentrenado.

b. Obtener el modelo de árbol de decisiones con los mejores parámetros que hayas encontrado con la métrica "precision" utilizada. Imprimir el valor de dicha métrica e incluye tus conclusiones finales para este caso.

In [ ]:
```
#Imprimimos los parámetros del mejor modelo
print('MODELO: ARBOL DE DECISION\n=======================\nMejores parámetros:\n')
print(mejores_params_DT)

#Entrenamos un nuevo clasificador con el mejor modelo
clf = mejor_modelo_DT.fit(X_train, Y_train)   # Entrenamos con el conjunto de prueba

#Usamos el mismo conjunto de entrenamiento para predecir que se obtendría
predi = clf.predict(X_train)

#Obtenemos el score utilizado para este modelo
print('Precisión en el conjunto de entrenamiento: %.2f' % precision(Y_train, predi))

#Ahora hacemos que el clasificador clasifique nuestro conjunto de pruebas que no ha
#usado hasta el momento.
predi = clf.predict(X_test)

#Finalmente obtenemos nuestro score con el conjunto de pruebas
print('Precisión en el conjunto de prueba: %.2f' % precision(Y_test, predi))
```

```
MODELO: ARBOL DE DECISION
=======================
Mejores parámetros:

{'ccp_alpha': 0.0, 'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 1,
'min_samples_split': 2}
Precisión en el conjunto de entrenamiento: 0.86
Precisión en el conjunto de prueba: 0.92
```

In [ ]:
```

```

Conclusión Árbol de Decisión

Comparando los modelos generados a partir del conjunto de entrenamiento, la curva de aprendizaje del modelo de parámetros mejorados nos indica que logramos optimizar el modelo, eliminando el sobreentrenamiento. Al evaluarlo con los datos de prueba corroboramos que se mantiene un buen desempeño e incluso se mejora pasando de una precisión del 86% al 92%.

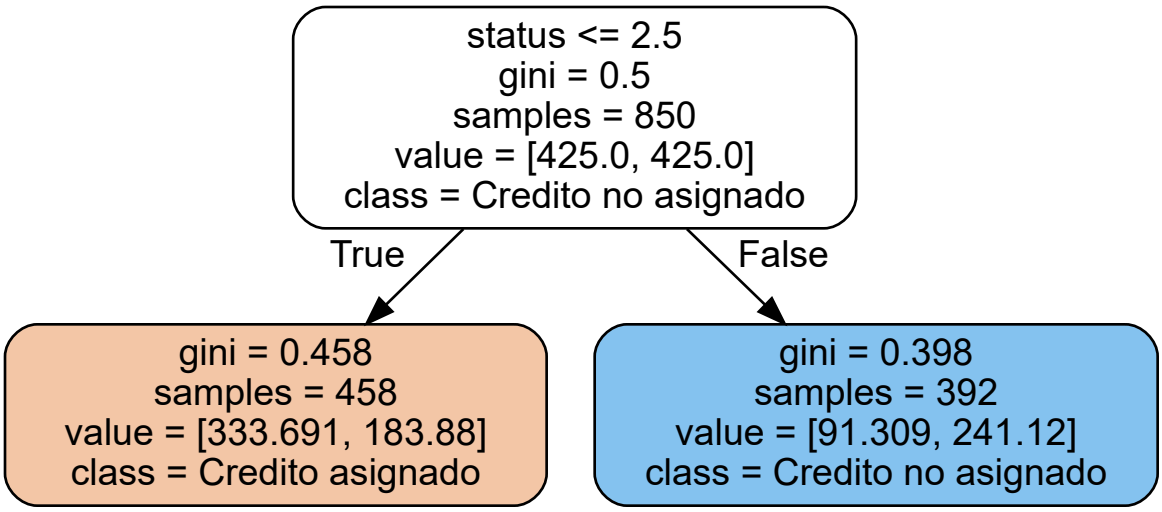
Esto se traduce a que mejoramos el modelo y no está sobreentrenado.

En el árbol, graficado a continuación, se puede observar un sólo nivel de profundidad lo cual incialmente nos causó dudas. Sinembargo, al analizarlo más en detalle, se observa que la variable de entrada utilizada para realizar el corte es 'status'. Esta variable clasifica los clientes de la siguiente manera: 1 : no checking account
2 : ... < 0 DM
3 : 0<= ... < 200 DM
4 : ... >= 200 DM / salary for at least 1 year

Esto nos revela que si el modelo vá a tomar la decisión basado en una sola variable, pues la mejor sería esta.

In [ ]:
```
export_graphviz(
    clf,
    out_file="miDT.dot",
    feature_names=list(pd.DataFrame(X_train).columns.values),
    class_names=['Credito asignado' , 'Credito no asignado'],
rounded=True,
filled=True
)

Source.from_file("miDT.dot")
```

Out[ ]:



c. Obtener el modelo de bosque aleatorio con los mejores parámetros que hayas encontrado con la métrica "recall" utilizada. Imprimir el valor de dicha métrica e incluye tus conclusiones finales para este caso.

In [ ]:
```
#Imprimimos los parámetros del mejor modelo
print('MODELO: BOSQUE ALEATORIO\n=======================\nMejores parámetros:\n')
print(mejores_params_RF)

#Entrenamos un nuevo clasificador con el mejor modelo
clf = mejor_modelo_RF.fit(X_train, Y_train)   # Entrenamos con el conjunto de prueba

#Usamos el mismo conjunto de entrenamiento para predecir que se obtendría
predi = clf.predict(X_train)
```

```
#Obtenemos el score utilizado para este modelo
print('Recall en el conjunto de entrenamiento: %.2f' % recall(Y_train, predi))

#Ahora hacemos que el clasificador clasifique nuestro conjunto de pruebas que no ha
#usado hasta el momento.
predi = clf.predict(X_test)

#Finalmente obtenemos nuestro score con el conjunto de pruebas
print('Recall en el conjunto de prueba: %.2f' % recall(Y_test, predi))
```

```
MODELO: BOSQUE ALEATORIO
========================
Mejores parámetros:

{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': 1, 'min_s
amples_split': 2}
Recall en el conjunto de entrenamiento: 1.00
Recall en el conjunto de prueba: 1.00
```

Conclusión Bosque aleatorio

El modelo sin ajustes de este algoritmo nos muestra un desempeño de sobre entrenamiento ya que la varianza es grande entre los resultados de prueba y validación.

Posteriormente al ajustar parámetros y obtener el score con recall , pareciera que mejoramos y que solucionamos el sobre entrenamiento del modelo. Sin embargo , debemos tener cuidado de no utilizar este score para determinar el desempeño del modelo ya que si evaluamos score mediante accuracy pareciera que el modelo está prediciendo a la clase mayoritaria e incorrectamente creer que optimizamos el modelo.

Finalmente entrenando al modelo con los parámetros optimizados con el conjunto de entrenamiento y haciendo predicciones de los datos de prueba, obtenemos resultados consistentes ya que el score se mantiene en un valor parecido al obtenido anteriormente.

En conclusión encontramos que para problemas con clases desbalanceadas debemos evaluar el desempeño con F1 score, recall y precisión

Bibliografía

Sadangi, S., (21 de Julio de 2022). How to Deal With Files in Google Colab: Everything You Need to Know. *Neptune Labs.* https://neptune.ai/blog/google-colab-dealing-with-files

Géron, A. (2022). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly Media, Inc.

scikit-learn.org. (s.f.). *sklearn.linear_model.LogisticRegression* scikit-learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html?highlight=logisticregression

scikit-learn.org. (s.f.). *Common pitfalls in the interpretation of coefficients of linear models.* scikit-learn.org. https://scikit-learn.org/stable/auto_examples/inspection/plot_linear_model_coefficient_interpretation.html

scikit-learn.org. (s.f.). *sklearn.model_selection.GridSearchCV* scikit-learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Fin de la Actividad de la semana 6.