Proyecto de clasificación con regresión logística en Python

José María Manzano Ortega

1 Preprocesamiento de los datos

En primer lugar, cargamos los datos de entrenamiento y test.

[29]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Lectura de los ficheros
data_train = pd.read_csv('anonymized-credit-rating/training_data.csv',␣
↪na_values=['?', 'NA', ''])
data_test = pd.read_csv('anonymized-credit-rating/test_data.csv', na_values=['?
↪', 'NA', ''])
```

[30]:
```
# Se muestran las primeras filas de los datos de entrenamiento
data_train.head()
```

[30]:

	ID	X1	X2	X3	X4	X5	X6	\
0	102	15502828	395185	2432028	10560043.0	25271554	2488.07	
1	343	142798094	41704189	55100678	87061505.0	182798367	36247.47	
2	407	42273308	9082199	26204830	18561405.0	32787980	15777.47	
3	950	37158119	2421374	6262830	19428247.0	33602057	6950.75	
4	848	20762774	2948439	15126377	10011602.0	10888065	4352.99	

	X7	X8	X9	...	X31	X32	X33	X34	X35	X36	\
0	4942785	112153.0	765.80	...	-1.53	1.35	1.98	2.08	12.25	10.21	
1	55736589	263186.0	626.02	...	13.59	14.67	25.12	82.71	78.71	18.30	
2	23711904	86666.0	194.01	...	11.62	5.87	12.87	35.68	84.53	12.17	
3	17729872	82913.0	294.75	...	18.02	6.56	11.75	20.89	16.13	20.88	
4	10751172	38454.0	213.49	...	12.24	9.19	19.05	40.57	80.46	21.84	

	X37	X38	X39	RATE
0	1.22	4.4206	-0.2186	B
1	10.71	1.8797	0.1190	B
2	11.09	1.1504	-0.0456	B
3	6.69	3.3327	0.0135	D
4	18.70	0.8287	0.0040	C

```
[5 rows x 41 columns]
```

[31]:
```
# Se muestran las primeras filas de los datos de test
data_test.head()
```

[31]:

	ID	X1	X2	X3	X4	X5	X6	\
0	656	144572752	10075885	47493866	NaN	113797217	29791.28	
1	174	17789687	12092136	14647400	12563356.0	7182412	5650.02	
2	129	53231078	24960422	39166500	20405047.0	17287176	9553.92	
3	674	42541535	20400239	21300464	10018069.0	29763977	5335.64	
4	197	84168684	44777986	49622098	59725046.0	40707971	13638.03	

	X7	X8	X9	...	X30	X31	X32	X33	X34	\
0	46876585	132674.0000	187.78	...	VTKGN	-3.94	8.63	13.09	236.49	
1	5226331	698147.5106	27.21	...	VTKGN	-5.20	4.32	6.12	22.43	
2	32826031	348441.0000	96.58	...	VTKGN	-0.39	0.92	2.55	1.50	
3	32523466	20647.0000	135.29	...	VTKGN	-2.11	-12.27	-45.79	-19.54	
4	24443638	129607.0000	299.32	...	VTKGN	13.40	6.24	8.70	65.82	

	X35	X36	X37	X38	X39
0	418.00	13.66	14.69	2.3756	0.1307
1	29.61	13.22	14.43	1.1511	0.0588
2	-46.86	30.91	-12.15	1.1019	0.0248
3	-62.08	3.54	-19.99	1.2343	-0.0815
4	65.95	22.07	16.79	1.4375	0.0922

```
[5 rows x 40 columns]
```

Las dimensiones de los conjuntos de entrenamiento y test son las siguientes:

[32]:
```
print(f"Conjunto de entrenamiento: {data_train.shape[0]} instancias de␣
 ↪{data_train.shape[1]} variables.")
print(f"Conjunto de test: {data_test.shape[0]} instancias de {data_test.
 ↪shape[1]} variables.")
```

```
Conjunto de entrenamiento: 906 instancias de 41 variables.
Conjunto de test: 389 instancias de 40 variables.
```

[33]:
```
# Columnas del conjunto de entrenamiento
data_train.columns
```

[33]:
```
Index(['ID', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10',
       'X11', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19', 'X20',
       'X21', 'X22', 'X23', 'X24', 'X25', 'X26', 'X27', 'X28', 'X29', 'X30',
       'X31', 'X32', 'X33', 'X34', 'X35', 'X36', 'X37', 'X38', 'X39', 'RATE'],
      dtype='object')
```

[34]:
```
# Estadística descriptiva
data_train.describe()
```

[34]:

	ID	X1	X2	X3	X4	\
count	906.000000	9.060000e+02	9.060000e+02	9.060000e+02	6.490000e+02	
mean	633.944812	3.448471e+08	1.409140e+08	2.296191e+08	1.679259e+08	
std	373.158757	1.011299e+09	4.666272e+08	7.688567e+08	5.322081e+08	
min	1.000000	4.871587e+06	6.290500e+04	6.277080e+05	1.918500e+04	
25%	314.500000	2.871623e+07	5.093032e+06	1.127887e+07	1.434103e+07	
50%	617.500000	5.905216e+07	1.601310e+07	2.742359e+07	3.163625e+07	
75%	956.750000	1.825440e+08	5.803176e+07	9.499336e+07	8.599445e+07	
max	1295.000000	7.254477e+09	3.540379e+09	5.955596e+09	4.242837e+09	

	X5	X6	X7	X8	X9	\
count	9.060000e+02	9.060000e+02	9.060000e+02	9.060000e+02	906.000000	
mean	3.226934e+08	7.095014e+04	1.834239e+08	2.789998e+06	427.673411	
std	9.703791e+08	2.430574e+05	5.380475e+08	1.856681e+07	413.669360	
min	1.279735e+06	-3.755140e+05	1.158190e+05	4.940000e+02	19.900000	
25%	2.302948e+07	3.216390e+03	1.071942e+07	9.595350e+04	177.312500	
50%	5.728552e+07	1.003438e+04	2.372510e+07	2.280075e+05	297.855000	
75%	1.707424e+08	3.388975e+04	7.744330e+07	6.547442e+05	511.952500	
max	6.869245e+09	2.000147e+06	3.313859e+09	2.679355e+08	2882.360000	

	...	X29	X31	X32	X33	X34	\
count	...	906.000000	906.000000	906.000000	906.000000	906.000000	
mean	...	1.652058	-2.698231	-0.769670	-6.350003	53.262748	
std	...	3.985377	118.955157	14.969628	49.989149	125.833097	
min	...	0.001800	-1291.330069	-59.883345	-390.835713	-405.450000	
25%	...	0.248725	-8.285000	-3.325000	-5.322500	-1.855000	
50%	...	0.754900	0.870000	2.665000	5.410000	33.670000	
75%	...	1.752250	5.030000	6.915000	12.762500	83.007500	
max	...	68.216900	1258.422867	57.725415	244.140000	787.504217	

	X35	X36	X37	X38	X39
count	906.000000	906.000000	906.000000	906.000000	906.000000
mean	33.762481	19.367833	-6.144963	1.868517	0.042568
std	152.764154	15.178570	37.857308	1.920547	0.272180
min	-716.824525	-31.590958	-192.898018	0.176400	-2.007800
25%	-19.915000	10.202500	-4.857500	1.019350	-0.033850
50%	24.490000	15.855000	3.355000	1.409100	0.064700
75%	79.115000	24.372500	8.807500	2.070850	0.155850
max	791.913297	69.717079	175.266141	33.602300	3.947700

[8 rows x 37 columns]

Identificamos los tipos de variables que tenemos, distinguiendo numéricas de categóricas.

[35]:
```
# Identifica las variables numéricas y las categóricas
numeric_features = data_train.select_dtypes(include=['int64', 'float64']).columns
categorical_features = data_train.select_dtypes(include=['object']).columns
```

[36]:
```
numeric_features
```

[36]:
```
Index(['ID', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10',
       'X11', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19', 'X20',
       'X21', 'X22', 'X23', 'X26', 'X27', 'X28', 'X29', 'X31', 'X32', 'X33',
       'X34', 'X35', 'X36', 'X37', 'X38', 'X39'],
      dtype='object')
```

[37]:
```
categorical_features
```

[37]:
```
Index(['X24', 'X25', 'X30', 'RATE'], dtype='object')
```

[38]:
```
# Obtiene los valores distintos de las columnas 'X24', 'X25', 'X30'
for variable in categorical_features:
    print(f"Valores distintos de {variable}:", data_train[variable].unique())
```

```
Valores distintos de X24: ['MED' 'HIGH' 'LOW' 'VHIGH' 'VLOW']
Valores distintos de X25: ['NO' 'YES']
Valores distintos de X30: ['VTKGN' 'XNHTQ' 'ASKVR' 'KUHMP' 'GXZVX' 'CLPXZ']
Valores distintos de RATE: ['B' 'D' 'C' 'A']
```

Los valores que toma la variable `X24` siguen un orden, desde "muy bajo" ("VLOW") hasta "muy alto" ("VHIGH"). La variable `X25` es binaria. Codificamos estas dos variables mediante *Ordinal Encoding*. Puesto que no conocemos si hay una relación ordinal entre los valores de `X30`, aplicaremos *One-Hot Encoding*. Creamos así un data frame únicamente variables numéricas para poder trabajar con clasificadores que no admitan variables categóricas.

[39]:
```
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder

# Eliminamos la variable objetivo y la variable ID
data_train_num = data_train.drop(["ID", "RATE"], axis = 1)
data_test_num = data_test.drop("ID", axis = 1)

# Valores de las variables
valores_X24 = ['VLOW', 'LOW', 'MED', 'HIGH', 'VHIGH']
valores_X25 = ['NO', 'YES']

# Aplica Ordinal Encoding a las variables X24 y X25
encoder_X24 = OrdinalEncoder(categories = [valores_X24], dtype=np.int32)
data_train_num[["X24"]] = encoder_X24.fit_transform(data_train_num[["X24"]])
data_test_num[["X24"]] = encoder_X24.transform(data_test_num[["X24"]])

encoder_X25 = OrdinalEncoder(categories = [valores_X25], dtype=np.int32)
data_train_num[["X25"]] = encoder_X25.fit_transform(data_train_num[["X25"]])
```

```
data_test_num[["X25"]] = encoder_X25.transform(data_test_num[["X25"]])

# Aplica OneHotEncoder a X30 y convierte el resultado a DataFrame
one_hot_encoder = OneHotEncoder(dtype=np.int32, sparse_output=False,␣
↪drop='first') # genera N-1 variables si hay N categorías distintas
encoded_X30_train = pd.DataFrame(one_hot_encoder.
↪fit_transform(data_train_num[['X30']]),
                                 columns=one_hot_encoder.
↪get_feature_names_out(['X30']))
encoded_X30_test = pd.DataFrame(one_hot_encoder.
↪transform(data_test_num[['X30']]),
                                columns=one_hot_encoder.
↪get_feature_names_out(['X30']))

# Asigna las columnas codificadas a los DataFrames
data_train_num[encoded_X30_train.columns] = encoded_X30_train
data_train_num.drop('X30', axis = 1, inplace=True)
data_test_num[encoded_X30_test.columns] = encoded_X30_test
data_test_num.drop('X30', axis = 1, inplace=True)
```

[40]:
```
print(data_train_num[['X30_CLPXZ', 'X30_GXZVX', 'X30_KUHMP', 'X30_VTKGN',␣
↪'X30_XNHTQ']])
```

```
     X30_CLPXZ  X30_GXZVX  X30_KUHMP  X30_VTKGN  X30_XNHTQ
0            0          0          0          1          0
1            0          0          0          1          0
2            0          0          0          1          0
3            0          0          0          1          0
4            0          0          0          1          0
..         ...        ...        ...        ...        ...
901          0          0          0          1          0
902          0          0          0          1          0
903          0          0          1          0          0
904          0          0          0          1          0
905          0          0          0          1          0

[906 rows x 5 columns]
```

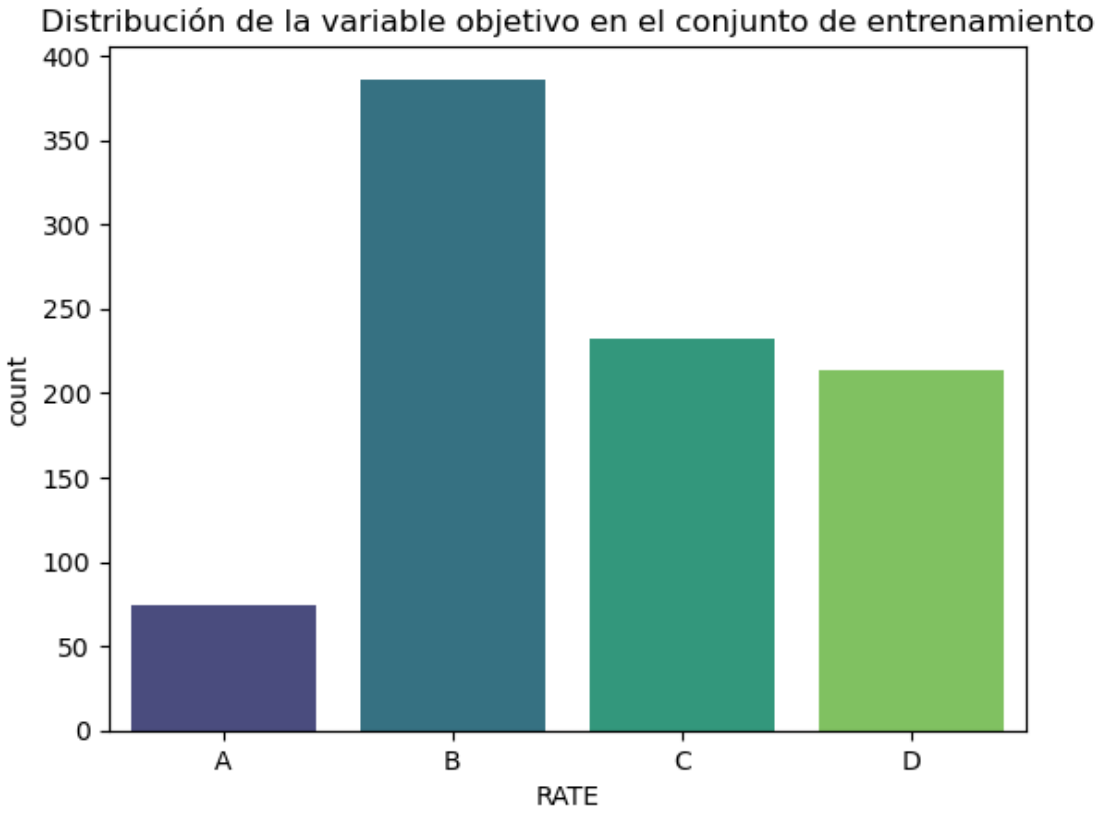
La variable objetivo es `RATE`. Estudiamos sus posibles valores.

[41]:
```
for clase in data_train['RATE'].unique():
    count_clase = data_train.value_counts('RATE')[clase]
    print(f"{clase}: {count_clase} ({count_clase / data_train.shape[0] * 100:.
↪1f}%)")
```

```
B: 386 (42.6%)
D: 214 (23.6%)
C: 232 (25.6%)
```

```
A: 74 (8.2%)
```

[42]:
```
import seaborn as sns

sns.countplot(x = 'RATE', data = data_train, order = ['A', 'B', 'C', 'D'],␣
↪palette = 'viridis')
plt.title('Distribución de la variable objetivo en el conjunto de entrenamiento')
plt.show()
```

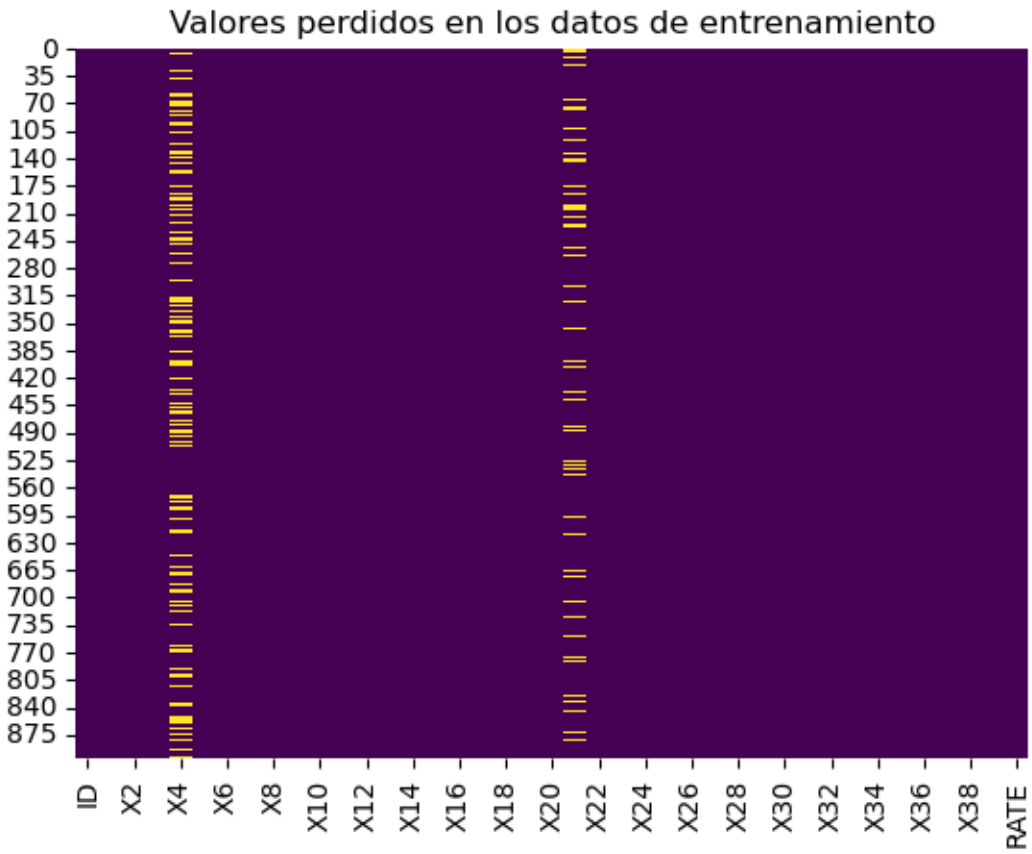


La variable `RATE` toma cuatro valores distintos, en distintas proporciones.

1.1 Tratamiento de valores perdidos

En primer lugar, comprobamos si existen valores perdidos en nuestros datos. Podemos visualizar las instancias con valores perdidos mediante el siguiente mapa de calor.

[43]:
```
sns.heatmap(data_train.isnull(), cmap='viridis', cbar=False)
plt.title("Valores perdidos en los datos de entrenamiento")
plt.show()
```



Observamos valores perdidos en las variables X4 y X21.

[44]:
```
# Columnas con valores perdidos
columnas_con_na = data_train.columns[data_train.isnull().any()]

# Calcular el porcentaje de valores perdidos en cada columna
numero_na = data_train[columnas_con_na].isnull().sum()
porcentaje_na = data_train[columnas_con_na].isnull().mean() * 100

# Mostrar las columnas con valores perdidos y su porcentaje de valores perdidos
for columna in columnas_con_na:
    print(f"Variable '{columna}': {numero_na[columna]} valores perdidos␣
 ↪({porcentaje_na[columna]:.2f}% de valores perdidos).")
```

```
Variable 'X4': 257 valores perdidos (28.37% de valores perdidos).
Variable 'X21': 121 valores perdidos (13.36% de valores perdidos).
```

Las variables X4 y X21 presentan un porcentaje relativamente alto de valores perdidos. Realizamos la misma comprobación en los datos de test.

[45]:
```
# Columnas con valores perdidos
columnas_con_na = data_test.columns[data_test.isnull().any()]

# Calcular el porcentaje de valores perdidos en cada columna
numero_na = data_test[columnas_con_na].isnull().sum()
porcentaje_na = data_test[columnas_con_na].isnull().mean() * 100

# Mostrar las columnas con valores perdidos y su porcentaje de valores perdidos
for columna in columnas_con_na:
    print(f"Variable '{columna}': {numero_na[columna]} valores perdidos␣
 ↪({porcentaje_na[columna]:.2f}% de valores perdidos).")
```

```
Variable 'X4': 111 valores perdidos (28.53% de valores perdidos).
Variable 'X21': 45 valores perdidos (11.57% de valores perdidos).
```

Las variables que presentan valores perdidos son las mismas que en los datos de entrenamiento, en proporción parecida.

[46]:
```
# Resumen descriptivo de las columnas con valores perdidos en entrenamiento
data_train[columnas_con_na].describe()
```

[46]:

	X4	X21
count	6.490000e+02	785.000000
mean	1.679259e+08	26.820789
std	5.322081e+08	17.791364
min	1.918500e+04	0.010000
25%	1.434103e+07	14.060000
50%	3.163625e+07	23.960000
75%	8.599445e+07	38.740000
max	4.242837e+09	83.971127

Comprobamos si existe relación entre la variable objetivo, `RATE`, y los valores perdidos.

[47]:
```
# Obtenemos las filas con valores perdidos
filas_con_na = data_train[data_train.isnull().any(axis=1)]

print("Filas con valores perdidos en alguna columna:")
filas_con_na[[*columnas_con_na, 'RATE']]
```

```
Filas con valores perdidos en alguna columna:
```

[47]:

	X4	X21	RATE
2	18561405.0	NaN	B
4	10011602.0	NaN	C
6	NaN	6.25	C
11	28245159.0	NaN	D
17	NaN	70.93	D
..	...	...	...
887	723642664.0	NaN	A

```
890          NaN   0.01    D
894          NaN  32.21    B
900          NaN  16.79    B
904          NaN   0.63    D

[345 rows x 3 columns]
```

[48]:
```
# Recuento de valores de la variable objetivo en las filas con valores perdidos
filas_con_na.value_counts('RATE')
```

[48]:
```
RATE
B    143
C     94
D     82
A     26
Name: count, dtype: int64
```

La proporción es las filas con valores perdidos que en el conjunto de entrenamiento completo. A continuación, realizamos una imputación de los valores perdidos utilizando el método KNN. Para ello, normalizamos antes los datos.

[49]:
```
from sklearn.preprocessing import StandardScaler

scaler_num = StandardScaler().fit(data_train_num)

# Normalización de los datos
data_train_num_scaled = pd.DataFrame(scaler_num.transform(data_train_num),␣
 ↪columns=data_train_num.columns)
data_test_num_scaled = pd.DataFrame(scaler_num.transform(data_test_num),␣
 ↪columns=data_test_num.columns)
```

[50]:
```
# Imputación KNN
from sklearn.impute import KNNImputer

knn_imp = KNNImputer(n_neighbors=4).fit(data_train_num_scaled)

# Imputación en entrenamiento
data_train_num_imp = pd.DataFrame(knn_imp.transform(data_train_num_scaled),␣
 ↪columns=data_train_num_scaled.columns)
data_train_num_imp.head()

# Imputación en test
data_test_num_imp = pd.DataFrame(knn_imp.transform(data_test_num_scaled),␣
 ↪columns=data_test_num_scaled.columns)
```

Comprobamos que todos los valores perdidos han sido imputados.

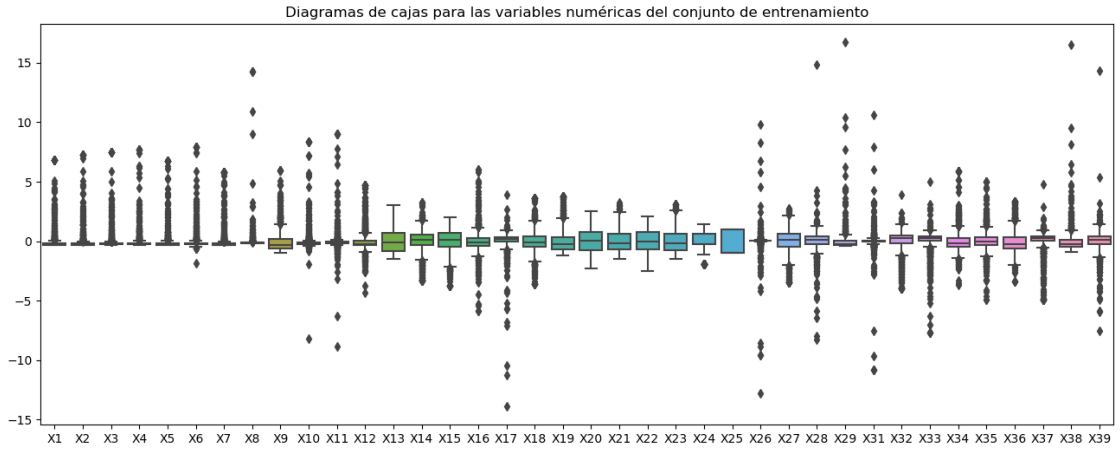
[51]:
```
# Columnas con valores perdidos
columnas_con_na_tras_imputar = data_train_num_imp.columns[data_train_num_imp.
 ↪isnull().any()]

# Calcular el porcentaje de valores perdidos en cada columna
numero_na = data_train_num_imp[columnas_con_na_tras_imputar].isnull().sum()
porcentaje_na = data_train_num_imp[columnas_con_na_tras_imputar].isnull().mean()␣
 ↪* 100

# Mostrar las columnas con valores perdidos y su porcentaje de valores perdidos
for columna in columnas_con_na_tras_imputar:
    print(f"Variable '{columna}': {numero_na[columna]} valores perdidos␣
 ↪({porcentaje_na[columna]:.2f}% de valores perdidos).")
else:
    print("Todos los valores perdidos han sido imputados.")
```

```
Todos los valores perdidos han sido imputados.
```

1.2 Tratamiento de *outliers*

Visualizamos los outliers con el método IQR a partir de diagramas de cajas.

[52]:
```
plt.figure(figsize=(16, 6))
sns.boxplot(data=data_train_num_imp.iloc[:,:38])
plt.title('Diagramas de cajas para las variables numéricas del conjunto de␣
 ↪entrenamiento')
plt.show()
```



Observamos una gran cantidad de variables con valores atípicos. Las variables de X1 hasta X10 presentan distribuciones con una larga cola a la derecha.

Emplearemos el algoritmo *Isolation Forest* para identificar outliers.

[53]:
```
from sklearn.ensemble import IsolationForest

# Detección de outliers con Isolation Forest
iso_forest = IsolationForest(n_estimators=100, random_state=42)
iso_forest.fit(data_train_num_imp)
data_train_num_outliers = data_train_num_imp.copy()
data_train_num_outliers["outlier"] = iso_forest.predict(data_train_num_outliers)
 ↪< 1
data_train_num_outliers.head(3)
```

[53]:
```
         X1        X2        X3        X4        X5        X6        X7  \
0 -0.325844 -0.301303 -0.295650 -0.295913 -0.306670 -0.281826 -0.331903
1 -0.199902 -0.212728 -0.227110 -0.152058 -0.144245 -0.142854 -0.237447
2 -0.299358 -0.282677 -0.264713 -0.280867 -0.298920 -0.227120 -0.297000

         X8        X9       X10  ...       X36       X37       X38       X39  \
0 -0.144307  0.817835 -0.239978  ... -0.603673  0.194653  1.329565 -0.960070
1 -0.136168  0.479746 -0.008804  ... -0.070390  0.445469  0.005826  0.280970
2 -0.145681 -0.565167 -0.223284  ... -0.474472  0.455513 -0.374119 -0.324111

   X30_CLPXZ  X30_GXZVX  X30_KUHMP  X30_VTKGN  X30_XNHTQ  outlier
0  -0.033241  -0.057639  -0.161393   0.214917  -0.088241    False
1  -0.033241  -0.057639  -0.161393   0.214917  -0.088241    False
2  -0.033241  -0.057639  -0.161393   0.214917  -0.088241    False

[3 rows x 44 columns]
```

[54]:
```
# Número de outliers
print("Número de outliers detectados:", data_train_num_outliers['outlier'].sum())

# Valores de 'RATE' para los outliers
data_train['RATE'][data_train_num_outliers[data_train_num_outliers['outlier'] ==
 ↪True].index].value_counts()
```

```
Número de outliers detectados: 54
```

[54]:
```
RATE
B    15
D    15
C    12
A    12
Name: count, dtype: int64
```

La proporción de los valores de las clases es distinta a la del conjunto de entrenamiento completo. La clase 'A' aparece aproximadamente en el 8% de los casos del conjunto completo, pero en el 22% de los outliers la variable objetivo toma este valor. Lo contrario ocurre con la clase 'B', que aparece en menor proporción en los outliers que en el conjunto total. Con solo 74 casos del valor 'A', no parece adecuado eliminar los outliers. Podemos disminuir el parámetro de contaminación

de Isolation Forest para reducir el número de outliers.

[55]:
```
from sklearn.ensemble import IsolationForest

# Detección de outliers con Isolation Forest
iso_forest = IsolationForest(n_estimators=100, random_state=42, contamination=0.
↪01)
iso_forest.fit(data_train_num_imp)
data_train_num_outliers = data_train_num_imp.copy()
data_train_num_outliers["outlier"] = iso_forest.predict(data_train_num_outliers)␣
↪< 1

# Número de outliers
print("Número de outliers detectados:", data_train_num_outliers['outlier'].sum())

# Valores de 'RATE' para los outliers
data_train['RATE'][data_train_num_outliers[data_train_num_outliers['outlier'] ==␣
↪True].index].value_counts()
```

```
Número de outliers detectados: 10
```

[55]:
```
RATE
A    7
B    2
D    1
Name: count, dtype: int64
```

Parece que la clase 'A' es la más inusual, en el sentido de que una parte significativa de las instancias que toman el valor 'A' en `RATE` poseen combinaciones de valores en las demás variables que permiten separarlas del resto más fácilmente.

1.3 Manejo de clases desbalanceadas

Hemos visto que no todas las clases de la variable objetivo `RATE` aparecen proporción similar en los datos. Las clases 'C' y 'D' aparecen cada una en la cuarta parte de las instancias. Sin embargo, la clase 'A' aparece solo en el 8.2% de los casos y la clase 'B', en el 42.6%. Realizaremos *oversampling* mediante el método SMOTE.

[56]:
```
from collections import Counter

# Almacenamos las variables predictoras en X_train y la variable objetivo en␣
↪y_train
X_train = data_train_num_imp
y_train = data_train['RATE']

print(X_train.shape)
print(sorted(Counter(y_train).items()))
```

```
(906, 43)
[('A', 74), ('B', 386), ('C', 232), ('D', 214)]
```

[57]:
```
# Oversampling mediante SMOTE
from imblearn.over_sampling import SMOTE
from collections import Counter

smote = SMOTE(random_state=42, k_neighbors=3)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
print(X_resampled.shape)
print(sorted(Counter(y_resampled).items()))
```

```
(1544, 43)
[('A', 386), ('B', 386), ('C', 386), ('D', 386)]
```

[58]:
```
from imblearn.over_sampling import SMOTE
from collections import Counter

smote = SMOTE(sampling_strategy={'A':250, 'B':386, 'C':250, 'D':250},␣
↪random_state=42, k_neighbors=3)
X_resampled_250, y_resampled_250 = smote.fit_resample(X_train, y_train)
print(X_resampled_250.shape)
print(sorted(Counter(y_resampled_250).items()))
```

```
(1136, 43)
[('A', 250), ('B', 386), ('C', 250), ('D', 250)]
```

A continuación realizamos una combinación de *over-sampling* y *under-sampling* con SMOTE seguido de ENN.

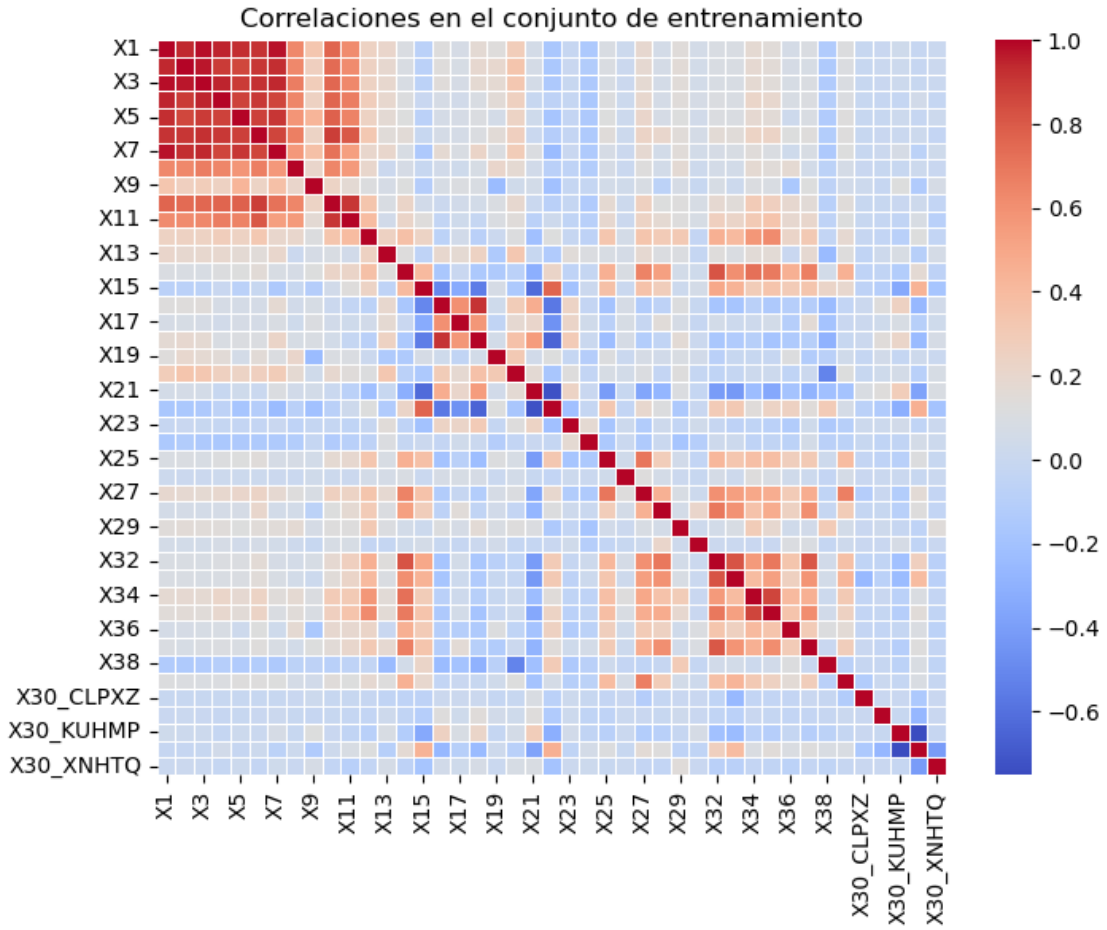
[59]:
```
# Oversampling + undersampling
from imblearn.combine import SMOTEENN
from imblearn.under_sampling import EditedNearestNeighbours
from collections import Counter

enn = EditedNearestNeighbours(sampling_strategy=['A','B', 'C', 'D'])

smoteenn = SMOTEENN(random_state=42, enn=enn)
X_resampled_smoteenn, y_resampled_smoteenn = smoteenn.fit_resample(X_train,␣
↪y_train)
print(X_resampled_smoteenn.shape)
print(sorted(Counter(y_resampled_smoteenn).items()))
```

```
(854, 43)
[('A', 371), ('B', 75), ('C', 159), ('D', 249)]
```

1.4 Reducción de la dimensionalidad

Representamos a continuación las correlaciones entre las variables.

[60]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(data_train_num_imp.corr(), cmap='coolwarm', fmt=".2f", linewidths=0.
↪5)
plt.title('Correlaciones en el conjunto de entrenamiento')
plt.show()
```

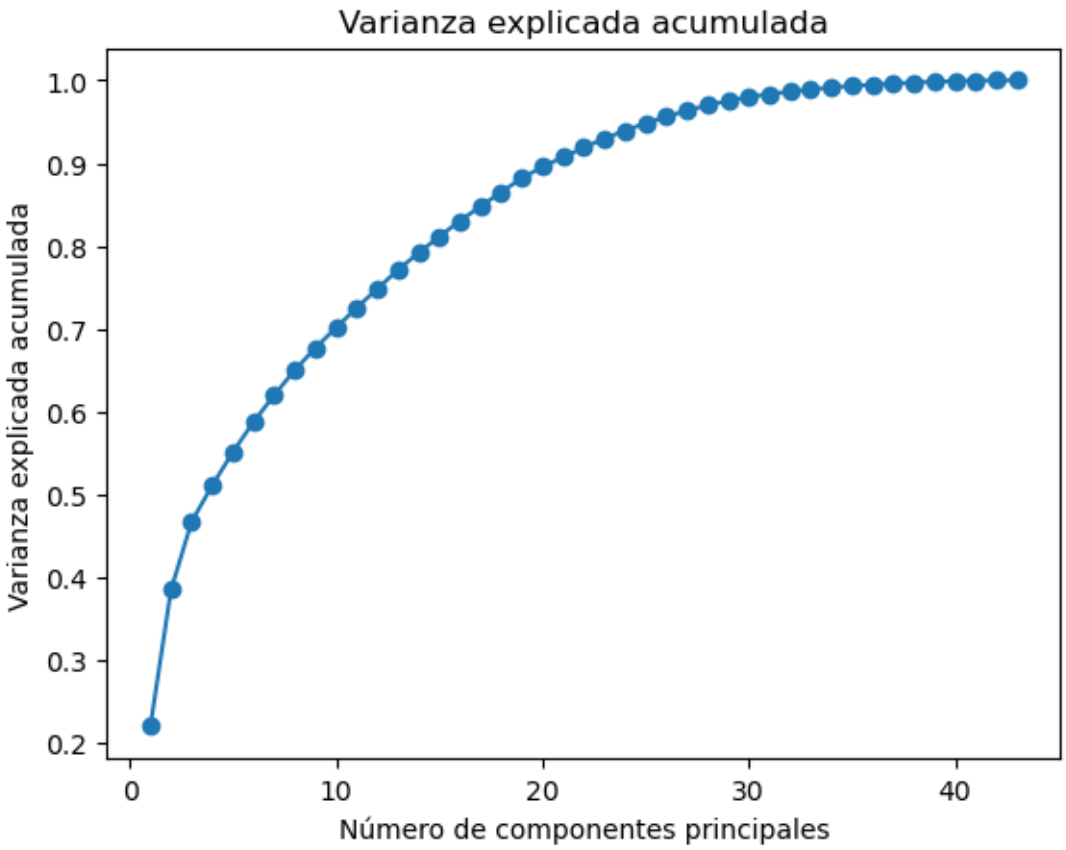


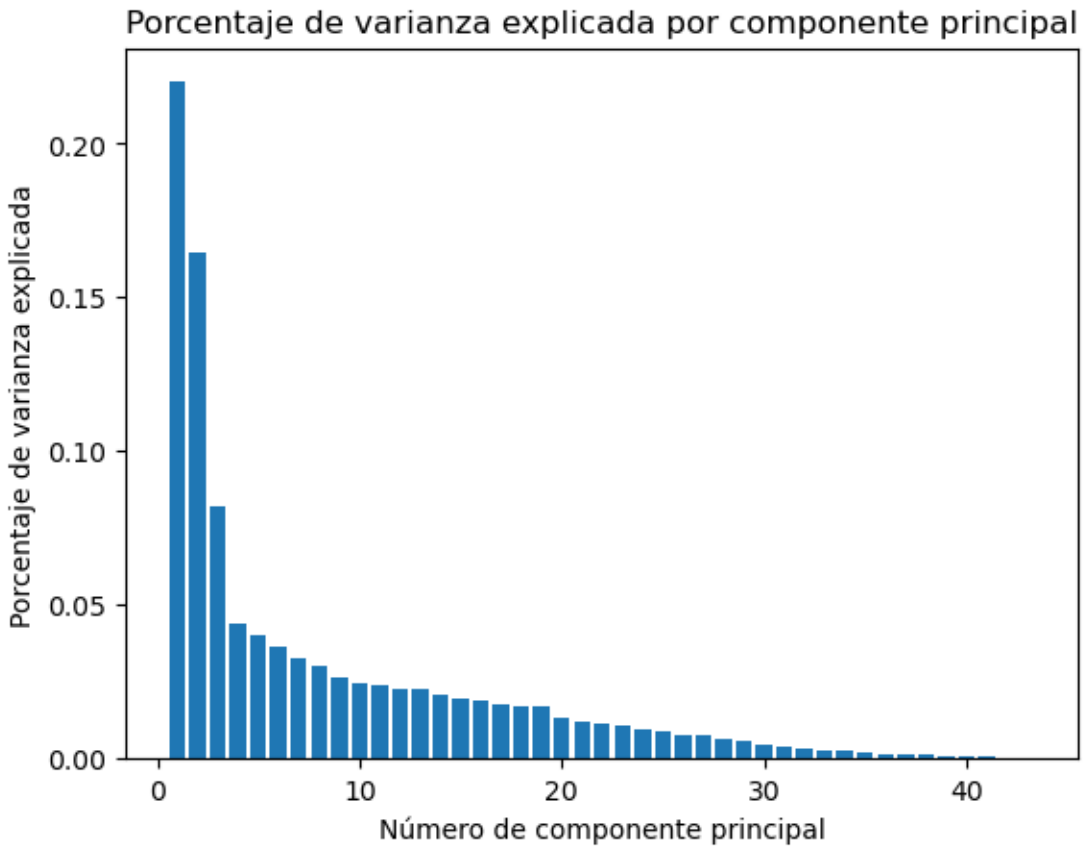
Observamos tres grupos de variables correladas positivamente entre sí; X1-X11, X16-X18 y el grupo formado por X12, X14, X15, X22, X25, X27, X28, X32-X37 y X39. Estas variables podrían aportar información redundante dentro de cada grupo sobre la clase, por lo que trataremos de reducir la dimensionalidad.

1.4.1 Análisis de Componentes Principales (PCA)

[61]:
```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

```
# Ajusta el modelo PCA
pca = PCA()
pca.fit(data_train_num_imp)

# Obtiene las componentes principales y sus proporciones de varianza explicada
principal_components = pca.components_
explained_variance_ratio = pca.explained_variance_ratio_

# Visualiza la varianza explicada acumulada
cumulative_explained_variance = explained_variance_ratio.cumsum()

plt.plot(range(1, len(cumulative_explained_variance) + 1),␣
 ↪cumulative_explained_variance, marker='o')
plt.title('Varianza explicada acumulada')
plt.xlabel('Número de componentes principales')
plt.ylabel('Varianza explicada acumulada')
plt.show()

# Grafica el porcentaje de varianza explicada por cada componente principal
plt.bar(range(1, len(explained_variance_ratio) + 1), explained_variance_ratio)
plt.title('Porcentaje de varianza explicada por componente principal')
plt.xlabel('Número de componente principal')
plt.ylabel('Porcentaje de varianza explicada')
plt.show()
```


Varianza explicada acumulada
Varianza explicada acumulada
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
10
20
30
40
Número de componentes principales



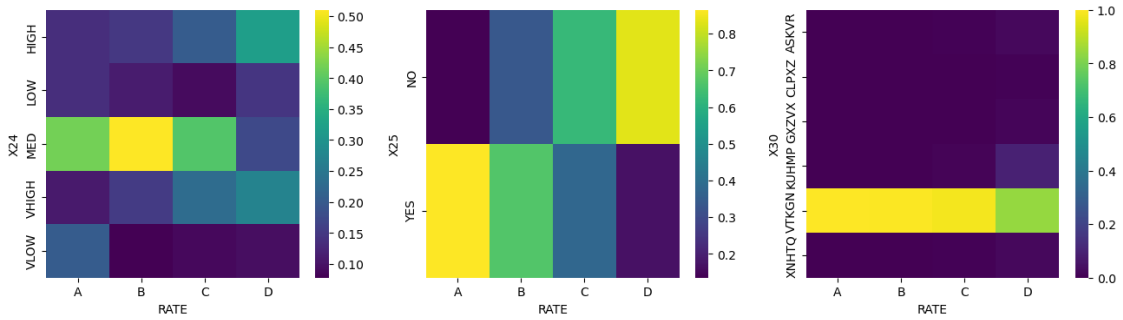
Para explicar el 90% de la varianza se necesitan 20 componentes. Exploramos otros métodos que nos permitan seleccionar características en función de la información que proporcionan sobre la variable `RATE`.

1.5 Gráficos

Los siguientes mapas de calor muestran para cada valor de `RATE` las proporciones en las que aparecen los valores de las tres variables categóricas.

[62]:
```
# Relación entre las variables categóricas y la variable objetivo
plt.figure(figsize=(16,4))
plt.subplot(1,3,1)
# Tabla de contingencia y mapa de calor
contingency_table = pd.crosstab(data_train['X24'], data_train['RATE'],␣
 ↪normalize='columns')
sns.heatmap(contingency_table, cmap='viridis')

plt.subplot(1,3,2)
contingency_table = pd.crosstab(data_train['X25'], data_train['RATE'],␣
 ↪normalize='columns')
```

```
sns.heatmap(contingency_table, cmap='viridis')

plt.subplot(1,3,3)
contingency_table = pd.crosstab(data_train['X30'], data_train['RATE'],␣
↪normalize='columns')
sns.heatmap(contingency_table, cmap='viridis')
plt.show()
```



Observamos que aproximadamente en la mitad de los casos en los que se da el valor 'B' en `RATE` se da también el valor 'MED' de `X24`. También tenemos que `X25` toma el valor 'YES' en un 90% de los casos en los que se da 'A', y el valor 'NO' casi siempre que se da 'D'. En cuanto a la variable `X30`, podemos ver que casi siempre toma el valor 'VTKGN' y cuando no, el valor de `RATE` suele ser 'D'.

2 Clasificación

Realizamos primero algunas pruebas sin utilizar selección de características. Trabajaremos con los datos normalizados, con valores perdidos imputados y, en algunos casos, con sobremuestreo.

2.1 Intento 1: Regresión logística

[ ]:
```
# Almacenamos las variables predictoras en X_train y la variable objetivo en␣
↪y_train
X_train = data_train_num_imp
y_train = data_train['RATE']

# Datos de test
X_test = data_test_num_imp
```

Comenzamos con regresión logística sin regularización.

[ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_predict, cross_val_score, KFold,␣
↪StratifiedKFold
from sklearn.metrics import confusion_matrix
```

```
# Modelo de regresión logística
model_logreg = LogisticRegression(penalty = None, max_iter=1000)

# Validación cruzada estratificada 10-fold
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
cross_val_scores = cross_val_score(model_logreg, X_train, y_train, cv=cv,␣
 ↪scoring='accuracy')
print("Accuracy:", np.mean(cross_val_scores))

# Predicciones en entrenamiento
y_train_pred = cross_val_predict(model_logreg, X_train, y_train, cv=cv)

# Matriz de confusión
conf_matrix = confusion_matrix(y_train, y_train_pred)
print("Matriz de Confusión:")
print(conf_matrix)
```

```
Accuracy: 0.6544078144078145
Matriz de Confusión:
[[ 34  39   1   0]
 [ 23 301  52  10]
 [  1  76 105  50]
 [  1  13  47 153]]
```

Lo repetimos eliminando las variables que presentaban valores perdidos.

[ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_predict, cross_val_score, KFold,␣
 ↪StratifiedKFold
from sklearn.metrics import confusion_matrix

# Modelo de regresión logística
model_logreg = LogisticRegression(penalty = None, max_iter=1000)

# Validación cruzada estratificada 10-fold
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
cross_val_scores = cross_val_score(model_logreg, X_train.drop(columnas_con_na,␣
 ↪axis=1), y_train, cv=cv, scoring='accuracy')
print("Accuracy:", np.mean(cross_val_scores))

# Predicciones en entrenamiento
y_train_pred = cross_val_predict(model_logreg, X_train.drop(columnas_con_na,␣
 ↪axis=1), y_train, cv=cv)

# Matriz de confusión
conf_matrix = confusion_matrix(y_train, y_train_pred)
print("Matriz de Confusión:")
print(conf_matrix)
```

```
Accuracy: 0.6466910866910865
Matriz de Confusión:
[[ 34  39   1   0]
 [ 25 296  57   8]
 [  1  79 101  51]
 [  1  12  46 155]]
```

Vemos que empeora el porcentaje de acierto al eliminar las variables que inicialmente tenían valores perdidos. Realizamos un Grid Search para encontrar una combinación de parámetros que mejore la tasa de acierto, considerando todas las variables.

[ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_predict, cross_val_score,␣
↪StratifiedKFold, GridSearchCV
from sklearn.metrics import confusion_matrix

# Modelo de regresión logística
model_logreg = LogisticRegression(max_iter=300)

# Validación cruzada estratificada 10-fold
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Grid Search
param_grid = {
    'C': [0.001, 0.01, 0.1, 1], # menores valores -> regularización más fuerte
    'class_weight': [None, 'balanced'], # aplica o no pesos a las clases
    'fit_intercept': [True, False] # incluye o no un término independiente
}
grid_search = GridSearchCV(model_logreg, param_grid, cv=cv, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Mejores hiperparámetros
print("Mejores hiperparámetros:", grid_search.best_params_)
```

```
Mejores hiperparámetros: {'C': 0.1, 'class_weight': None, 'fit_intercept': True}
```

[ ]:
```
# Accuracy del modelo
cross_val_scores = cross_val_score(grid_search, X_train, y_train, cv=cv,␣
↪scoring='accuracy')
print("Accuracy:", np.mean(cross_val_scores))

# Predicciones en entrenamiento
y_train_pred = grid_search.predict(X_train)

# Matriz de confusión
conf_matrix = confusion_matrix(y_train, y_train_pred)
print("Matriz de Confusión:")
print(conf_matrix)
```

```
Accuracy: 0.6567399267399268
Matriz de Confusión:
[[ 29  45   0   0]
 [  9 335  34   8]
 [  2  82 105  43]
 [  0  11  30 173]]
```

Aplicamos el clasificador a los datos de test.

[ ]:
```
y_test_pred = grid_search.predict(X_test)

# Número de instancias etiquetadas en cada clase
print(pd.Series(y_test_pred).value_counts())
```

```
B    201
D     95
C     81
A     12
Name: count, dtype: int64
```

[ ]:
```
# Resultados con el formato de entrega para Kaggle
results1 = pd.DataFrame({'ID': data_test['ID'], 'RATE': y_test_pred})
print(results1.head())
print(results1.shape)

results1.to_csv('results1.csv', index=False)
```

```
    ID RATE
0  656    B
1  174    B
2  129    C
3  674    D
4  197    B
(389, 2)
```

2.2 Intento 2: Oversampling con SMOTE + Regresión Logística

Trabajamos con los datos normalizados, con valores perdidos imputados y sobremuestreo con SMOTE aplicado.

[38]:
```
X_train = X_resampled
y_train = y_resampled

X_test = data_test_num_imp
```

[174]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_predict, cross_val_score,␣
 ↪StratifiedKFold, GridSearchCV
from sklearn.metrics import confusion_matrix
```

```
# Modelo de regresión logística
model_logreg = LogisticRegression(max_iter=300)

# Validación cruzada estratificada 10-fold
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Grid Search
param_grid = {
    'C': [0.001, 0.01, 0.1, 1], # menores valores -> regularización más fuerte
    'class_weight': [None, 'balanced'], # aplica o no pesos a las clases
    'fit_intercept': [True, False] # incluye o no un término independiente
}
grid_search = GridSearchCV(model_logreg, param_grid, cv=cv, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Mejores hiperparámetros
print("Mejores hiperparámetros:", grid_search.best_params_)
```

```
Mejores hiperparámetros: {'C': 1, 'class_weight': None, 'fit_intercept': True}
```

[175]:
```
# Accuracy del modelo
cross_val_scores = cross_val_score(grid_search, X_train, y_train, cv=cv,␣
 ↪scoring='accuracy')
print("Accuracy:", np.mean(cross_val_scores))

# Predicciones en entrenamiento
y_train_pred = grid_search.predict(X_train)

# Matriz de confusión
conf_matrix = confusion_matrix(y_train, y_train_pred)
print("Matriz de Confusión:")
print(conf_matrix)
```

```
Accuracy: 0.7111059907834102
Matriz de Confusión:
[[345  41   0   0]
 [ 65 228  86   7]
 [  4  59 265  58]
 [  2   7  57 320]]
```

Evaluamos ahora el modelo en el conjunto de entrenamiento sin oversampling.

[176]:
```
from sklearn.metrics import accuracy_score
y_train_pred = grid_search.predict(data_train_num_imp)

print("Accuracy:", accuracy_score(data_train['RATE'], y_train_pred))
```

```
# Matriz de confusión
conf_matrix = confusion_matrix(data_train['RATE'], y_train_pred)
print("Matriz de Confusión:")
print(conf_matrix)
```

```
Accuracy: 0.6666666666666666
Matriz de Confusión:
[[ 62  12   0   0]
 [ 65 228  86   7]
 [  4  38 144  46]
 [  1   6  37 170]]
```

Este clasificador parece funcionar mejor que el anterior con la clase A y la C, pero peor con la B. Aplicamos el modelo al conjunto de test.

[177]:
```
y_test_pred = grid_search.predict(X_test)

# Número de instancias etiquetadas en cada clase
print(pd.Series(y_test_pred).value_counts())
```

```
B    135
C    113
D     97
A     44
Name: count, dtype: int64
```

[178]:
```
# Resultados con el formato de entrega para Kaggle
results2 = pd.DataFrame({'ID': data_test['ID'], 'RATE': y_test_pred})
print(results2.head())
print(results2.shape)

results2.to_csv('results2.csv', index=False)
```

```
    ID RATE
0  656    A
1  174    B
2  129    D
3  674    D
4  197    B
(389, 2)
```

2.2.1 Utilizando SMOTE con menor cantidad de instancias generadas

Como la capacidad para clasificar correctamente la clase 'B' se ha visto prejudicada por SMOTE, tratamos de evitarlo utilizando menor cantidad instancias generadas (hasta 250 de cada clase).

[67]:
```
X_train = X_resampled_250
y_train = y_resampled_250
```

```
X_test = data_test_num_imp


from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_predict, cross_val_score,␣
↪StratifiedKFold, GridSearchCV
from sklearn.metrics import confusion_matrix

# Modelo de regresión logística
model_logreg = LogisticRegression(max_iter=300)

# Validación cruzada estratificada 10-fold
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Grid Search
param_grid = {
    'C': [0.001, 0.01, 0.1, 1], # menores valores -> regularización más fuerte
    'class_weight': [None, 'balanced'], # aplica o no pesos a las clases
    'fit_intercept': [True, False] # incluye o no un término independiente
}
grid_search = GridSearchCV(model_logreg, param_grid, cv=cv, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Mejores hiperparámetros
print("Mejores hiperparámetros:", grid_search.best_params_)
```

```
Mejores hiperparámetros: {'C': 0.1, 'class_weight': None, 'fit_intercept': True}
```

[68]:
```
# Evaluación del modelo en el conjunto de entrenamiento
from sklearn.metrics import accuracy_score
y_train_pred = grid_search.predict(data_train_num_imp)

print("Accuracy:", accuracy_score(data_train['RATE'], y_train_pred))

# Matriz de confusión
conf_matrix = confusion_matrix(data_train['RATE'], y_train_pred)
print("Matriz de Confusión:")
print(conf_matrix)
```

```
Accuracy: 0.6843267108167771
Matriz de Confusión:
[[ 50  24   0   0]
 [ 49 289  40   8]
 [  3  77 106  46]
 [  1   7  31 175]]
```

Como esperábamos, este modelo es más equilibrado que el anterior.

2.3 Intento 3: Bagging con el clasificador del Intento 1

Se aplicará bagging (implementado en el fichero `bagging.py`) con 10 estimadores al clasificador anterior.

[29]:
```
# Almacenamos las variables predictoras en X_train y la variable objetivo en␣
↪y_train
X_train = data_train_num_imp
y_train = data_train['RATE']

# Datos de test
X_test = data_test_num_imp
```

[30]:
```
from bagging import Bagging

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix, accuracy_score

# Modelo de regresión logística
model_logreg = LogisticRegression(C=0.1, class_weight=None, fit_intercept=True,␣
↪max_iter=300)

# Entrenamiento del modelo con Bagging
bag = Bagging([model_logreg], 10, random_state=42)

# Validación cruzada estratificada 10-fold
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
accs = []
for train_index, test_index in kf.split(X_train, y_train):
    X_train_fold = X_train.iloc[train_index]
    X_test_fold = X_train.iloc[test_index]
    y_train_fold = y_train[train_index]
    y_test_fold = y_train[test_index]

    # Entrenamos y evaluamos cada fold
    bag.fit(X_train_fold, y_train_fold)
    accs.append(accuracy_score(y_test_fold, bag.predict(X_test_fold)))

print("Accuracy en validación cruzada:", np.mean(accs))

# Predicciones en entrenamiento
bag.fit(X_train, y_train)
y_train_pred = bag.predict(X_train)
print("Accuracy en entrenamiento:", accuracy_score(y_train, y_train_pred))

# Matriz de confusión
conf_matrix = confusion_matrix(y_train, y_train_pred)
```

```
print("Matriz de Confusión:")
print(conf_matrix)
```

```
Accuracy en validación cruzada: 0.6357753357753356
Accuracy en entrenamiento: 0.6843267108167771
Matriz de Confusión:
[[ 32  41   1   0]
 [ 19 326  29  12]
 [  2  89  91  50]
 [  0   9  34 171]]
```

[32]:
```
y_test_pred = bag.predict(X_test)

# Número de instancias etiquetadas en cada clase
print(pd.Series(y_test_pred).value_counts())
```

```
0
B    203
D    112
C     59
A     15
Name: count, dtype: int64
```

[38]:
```
# Resultados con el formato de entrega para Kaggle
results3 = pd.DataFrame({'ID': data_test['ID'], 'RATE': y_test_pred})

results3.to_csv('results3.csv', index=False)
```

2.4 Intento 4: Bagging y SMOTE

Probamos bagging con SMOTE.

[31]:
```
# Almacenamos las variables predictoras en X_train y la variable objetivo en␣
↪y_train
X_train = X_resampled
y_train = y_resampled

# Datos de test
X_test = data_test_num_imp
```

[32]:
```
from bagging import Bagging

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix, accuracy_score

# Modelo de regresión logística
```

```
model_logreg = LogisticRegression(C=0.1, class_weight=None, fit_intercept=True,␣
 ↪max_iter=300)

# Entrenamiento del modelo con Bagging
bag = Bagging([model_logreg], 10, random_state=42)

# Validación cruzada estratificada 10-fold
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
accs = []
for train_index, test_index in kf.split(data_train_num_imp, data_train['RATE']):
    X_train_fold = X_train.iloc[train_index]
    X_test_fold = X_train.iloc[test_index]
    y_train_fold = y_train[train_index]
    y_test_fold = y_train[test_index]

    # Entrenamos y evaluamos cada fold
    bag.fit(X_train_fold, y_train_fold)
    accs.append(accuracy_score(y_test_fold, bag.predict(X_test_fold)))

print("Accuracy en validación cruzada:", np.mean(accs))

# Predicciones en entrenamiento
bag.fit(X_train, y_train)
y_train_pred = bag.predict(data_train_num_imp)
print("Accuracy en entrenamiento:", accuracy_score(data_train['RATE'],␣
 ↪y_train_pred))

# Matriz de confusión
conf_matrix = confusion_matrix(data_train['RATE'], y_train_pred)
print("Matriz de Confusión:")
print(conf_matrix)
```

```
Accuracy en validación cruzada: 0.6357753357753356
Accuracy en entrenamiento: 0.6169977924944813
Matriz de Confusión:
[[ 59  14   1   0]
 [ 71 192 113  10]
 [  6  41 138  47]
 [  2   5  37 170]]
```

[46]:
```
y_test_pred = bag.predict(X_test)

# Número de instancias etiquetadas en cada clase
print(pd.Series(y_test_pred).value_counts())
```

```
0
C    131
B    107
```

```
D     96
A     55
Name: count, dtype: int64
```

[47]:
```
# Resultados con el formato de entrega para Kaggle
results4 = pd.DataFrame({'ID': data_test['ID'], 'RATE': y_test_pred})

results4.to_csv('results4.csv', index=False)
```

2.4.1 Utilizando SMOTE con menor número de instancias generadas

[69]:
```
# Almacenamos las variables predictoras en X_train y la variable objetivo en␣
 ↪y_train
X_train = X_resampled_250
y_train = y_resampled_250

# Datos de test
X_test = data_test_num_imp

from bagging import Bagging

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix, accuracy_score

# Modelo de regresión logística
model_logreg = LogisticRegression(C=0.1, class_weight=None, fit_intercept=True,␣
 ↪max_iter=300)

# Entrenamiento del modelo con Bagging
bag = Bagging([model_logreg], 10, random_state=42)

# Validación cruzada estratificada 10-fold
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
accs = []
for train_index, test_index in kf.split(data_train_num_imp, data_train['RATE']):
    X_train_fold = X_train.iloc[train_index]
    X_test_fold = X_train.iloc[test_index]
    y_train_fold = y_train[train_index]
    y_test_fold = y_train[test_index]

    # Entrenamos y evaluamos cada fold
    bag.fit(X_train_fold, y_train_fold)
    accs.append(accuracy_score(y_test_fold, bag.predict(X_test_fold)))

print("Accuracy en validación cruzada:", np.mean(accs))
```

```
# Predicciones en entrenamiento
bag.fit(X_train, y_train)
y_train_pred = bag.predict(data_train_num_imp)
print("Accuracy en entrenamiento:", accuracy_score(data_train['RATE'],␣
↪y_train_pred))

# Matriz de confusión
conf_matrix = confusion_matrix(data_train['RATE'], y_train_pred)
print("Matriz de Confusión:")
print(conf_matrix)
```

```
Accuracy en validación cruzada: 0.6357753357753356
Accuracy en entrenamiento: 0.6766004415011038
Matriz de Confusión:
[[ 53  21   0   0]
 [ 48 278  51   9]
 [  5  64 118  45]
 [  2   9  39 164]]
```

2.5 Intento 5: Regularización con Elastic Net

[63]:
```
# Datos de entrenamiento
X_train = data_train_num_imp
y_train = data_train['RATE']

# Datos de test
X_test = data_test_num_imp
```

[64]:
```
import warnings
warnings.filterwarnings("ignore")
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.model_selection import StratifiedKFold, GridSearchCV


# Modelo de regresión logística
model_logreg = LogisticRegression(multi_class='multinomial',␣
↪penalty='elasticnet', solver='saga',
                                  max_iter=300, tol=0.001, l1_ratio = 0.5,␣
↪random_state=42)

# pipeline PCA + Regresión logística
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA()),
    ('logistic_regression', model_logreg)
```

```
])

# Define el espacio de búsqueda de parámetros
param_grid = {
    'pca__n_components': [35, 37, 40],
    'logistic_regression__multi_class': ['ovr', 'multinomial'],
    'logistic_regression__C': [0.1, 0.2, 0.25],
    'logistic_regression__l1_ratio': [.01, .1, .2, .5, 1]
}

# Validación cruzada estratificada 10-fold
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Búsqueda de hiperparámetros
grid_search = GridSearchCV(pipeline, param_grid, cv=kf, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Mejores hiperparámetros
print("Mejores hiperparámetros:", grid_search.best_params_)
```

```
Mejores hiperparámetros: {'logistic_regression__C': 0.2,
'logistic_regression__l1_ratio': 0.01, 'logistic_regression__multi_class':
'multinomial', 'pca__n_components': 37}
```

[148]:
```
from bagging import Bagging

from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix, accuracy_score

# Entrenamiento del modelo con Bagging
bag = Bagging([grid_search.best_estimator_], 50, random_state=42)

# Validación cruzada estratificada 10-fold
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
accs = []
for train_index, test_index in kf.split(data_train_num_imp, data_train['RATE']):
    X_train_fold = X_train.iloc[train_index]
    X_test_fold = X_train.iloc[test_index]
    y_train_fold = y_train[train_index]
    y_test_fold = y_train[test_index]

    # Entrenamos y evaluamos cada fold
    bag.fit(X_train_fold, y_train_fold)
    accs.append(accuracy_score(y_test_fold, bag.predict(X_test_fold)))

print("Accuracy en validación cruzada:", np.mean(accs))
```

```
# Predicciones en entrenamiento
bag.fit(X_train, y_train)
y_train_pred = bag.predict(data_train_num_imp)
print("Accuracy en entrenamiento:", accuracy_score(data_train['RATE'],␣
 ↪y_train_pred))

# Matriz de confusión
conf_matrix = confusion_matrix(data_train['RATE'], y_train_pred)
print("Matriz de Confusión:")
print(conf_matrix)
```

```
Accuracy en validación cruzada: 0.6146642246642247
Accuracy en entrenamiento: 0.6821192052980133
Matriz de Confusión:
[[ 34  40   0   0]
 [ 11 337  29   9]
 [  0 103  70  59]
 [  0  15  22 177]]
```

Probamos a utilizar los datos con sobremuestreo.

[150]:
```
X_train = X_resampled_250
y_train = y_resampled_250

X_test = data_test_num_imp

# Grid Search para los datos con sobremuestreo
grid_search.fit(X_train, y_train)

bag = Bagging([grid_search.best_estimator_], 50, random_state=42)

# Bagging con validación cruzada
accs = []
for train_index, test_index in kf.split(data_train_num_imp, data_train['RATE']):
    X_train_fold = X_train.iloc[train_index]
    X_test_fold = X_train.iloc[test_index]
    y_train_fold = y_train[train_index]
    y_test_fold = y_train[test_index]

    # Entrenamos y evaluamos cada fold
    bag.fit(X_train_fold, y_train_fold)
    accs.append(accuracy_score(y_test_fold, bag.predict(X_test_fold)))

print("Accuracy en validación cruzada:", np.mean(accs))

# Predicciones en entrenamiento
bag.fit(X_train, y_train)
y_train_pred = bag.predict(data_train_num_imp)
```

```
print("Accuracy en entrenamiento:", accuracy_score(data_train['RATE'],␣
 ↪y_train_pred))

# Matriz de confusión
conf_matrix = confusion_matrix(data_train['RATE'], y_train_pred)
print("Matriz de Confusión:")
print(conf_matrix)
```

```
c:\Users\jose\anaconda3\envs\mineriai\lib\site-
packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
c:\Users\jose\anaconda3\envs\mineriai\lib\site-
packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
c:\Users\jose\anaconda3\envs\mineriai\lib\site-
packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
```

```
Accuracy en validación cruzada: 0.6146642246642247
Accuracy en entrenamiento: 0.6810154525386314
Matriz de Confusión:
[[ 50  23   1   0]
 [ 44 296  30  16]
 [  2  79  93  58]
 [  1  10  25 178]]
```

Añadimos selección de características y lo probamos con el conjunto de datos con SMOTE+ENN aplicado.

[222]:
```
X_train = X_resampled_smoteenn
y_train = y_resampled_smoteenn

X_test = data_test_num_imp

import warnings
warnings.filterwarnings("ignore")
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.feature_selection import SelectFromModel

# Modelo de regresión logística
model_logreg = LogisticRegression(multi_class='multinomial',␣
 ↪penalty='elasticnet', solver='saga',
```

```
                                    max_iter=300, tol=0.001, l1_ratio = 0.5,␣
↪random_state=42)

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA()),
    ('feature_selection', SelectFromModel(model_logreg, threshold='0.5*mean')),
    ('logistic_regression', model_logreg)
])

# Define el espacio de búsqueda de parámetros
param_grid = {
    'pca__n_components': [35, 37, 40],
    'logistic_regression__multi_class': ['ovr', 'multinomial'],
    'logistic_regression__C': [0.01, 0.1],
    'logistic_regression__l1_ratio': [0.1, 0.5, 0.9]
}

# Validación cruzada estratificada 10-fold
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Búsqueda de hiperparámetros
grid_search = GridSearchCV(pipeline, param_grid, cv=kf, scoring='accuracy')
grid_search.fit(X_train, y_train)
print("Mejores hiperparámetros:", grid_search.best_params_)
```

```
Mejores hiperparámetros: {'logistic_regression__C': 0.1,
'logistic_regression__l1_ratio': 0.1, 'logistic_regression__multi_class':
'multinomial', 'pca__n_components': 35}
```

[224]:
```
import warnings
warnings.filterwarnings("ignore")

bag = Bagging([grid_search.best_estimator_], 50, random_state=42)

kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Bagging con validación cruzada
accs = []
for train_index, test_index in kf.split(X_train, y_train):
    X_train_fold = X_train.iloc[train_index]
    X_test_fold = X_train.iloc[test_index]
    y_train_fold = y_train[train_index]
    y_test_fold = y_train[test_index]

    # Entrenamos y evaluamos cada fold
    bag.fit(X_train_fold, y_train_fold)
```

```
    accs.append(accuracy_score(y_test_fold, bag.predict(X_test_fold)))

print("Accuracy en validación cruzada:", np.mean(accs))

# Predicciones en entrenamiento
bag.fit(X_train, y_train)
y_train_pred = bag.predict(data_train_num_imp)
print("Accuracy en entrenamiento:", accuracy_score(data_train['RATE'],␣
 ↪y_train_pred))

# Matriz de confusión
conf_matrix = confusion_matrix(data_train['RATE'], y_train_pred)
print("Matriz de Confusión:")
print(conf_matrix)
```

```
Accuracy en validación cruzada: 0.8688372093023256
Accuracy en entrenamiento: 0.5
Matriz de Confusión:
[[ 70   1   3   0]
 [138  84 139  25]
 [ 19  19 132  62]
 [  5   2  40 167]]
```

[170]:
```
y_test_pred = bag.predict(X_test)

# Número de instancias etiquetadas en cada clase
print(pd.Series(y_test_pred).value_counts())
```

```
0
C    135
B    118
D     83
A     53
Name: count, dtype: int64
```

[181]:
```
X_train = data_train_num_imp
y_train = data_train['RATE']

from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.feature_selection import SelectFromModel

# Validación cruzada estratificada 10-fold
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Modelo de regresión logística
```

```
model_logreg = LogisticRegression(multi_class='multinomial', solver='lbfgs',␣
 ↪max_iter=300, random_state=42)

# Selección de características
sel = SelectFromModel(model_logreg, threshold='0.5*mean')
sel.fit(X_train, y_train)

# Atributos seleccionados
selected_features = X_train.columns[sel.get_support()]
print("Número de características seleccionadas:", len(selected_features))
print("Características seleccionadas:", selected_features)

# Conjuntos de entrenamiento y test tras selección de características y PCA para␣
 ↪agrupar
# las variables relevantes que presentan correlaciones elevadas
pca = PCA(n_components=20)
X_train_sel = X_train[selected_features]
X_test_sel = X_test[selected_features]

# Grid Search
param_grid = {
    'C': [0.01, 0.1, 0.5, 1], # menores valores -> regularización más fuerte
    'class_weight': [None, 'balanced'], # aplica o no pesos a las clases
}
grid_search = GridSearchCV(model_logreg, param_grid, cv=kf, scoring='accuracy')
grid_search.fit(X_train_sel, y_train)

# Mejores hiperparámetros
print("Mejores hiperparámetros:", grid_search.best_params_)
```

```
Número de características seleccionadas: 30
Características seleccionadas: Index(['X1', 'X2', 'X3', 'X4', 'X5', 'X7', 'X8',
'X10', 'X13', 'X14', 'X15',
       'X16', 'X18', 'X20', 'X21', 'X22', 'X23', 'X24', 'X27', 'X28', 'X32',
       'X33', 'X34', 'X35', 'X36', 'X37', 'X38', 'X39', 'X30_KUHMP',
       'X30_VTKGN'],
      dtype='object')
Mejores hiperparámetros: {'C': 0.5, 'class_weight': None}
```

Las características seleccionadas son las que contribuyen más a la regresión logística (las que tienen un coeficiente mayor que la cuarta parte de la media de los coeficientes del modelo).

[182]:
```
from bagging import Bagging

# Entrenamiento del modelo con Bagging
bag = Bagging([grid_search.best_estimator_], 50, random_state=42)

# Validación cruzada estratificada 10-fold
```

```
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
accs = []
for train_index, test_index in kf.split(X_train_sel, y_train):
    X_train_fold = X_train_sel.iloc[train_index]
    X_test_fold = X_train_sel.iloc[test_index]
    y_train_fold = y_train[train_index]
    y_test_fold = y_train[test_index]

    # Entrenamos y evaluamos cada fold
    bag.fit(X_train_fold, y_train_fold)
    accs.append(accuracy_score(y_test_fold, bag.predict(X_test_fold)))

print("Accuracy en validación cruzada:", np.mean(accs))

# Predicciones en entrenamiento
bag.fit(data_train_num_imp[selected_features], data_train['RATE'])
y_train_pred = bag.predict(X_train_sel)
print("Accuracy en entrenamiento:", accuracy_score(y_train, y_train_pred))

# Matriz de confusión
conf_matrix = confusion_matrix(y_train, y_train_pred)
print("Matriz de Confusión:")
print(conf_matrix)
```

```
Accuracy en validación cruzada: 0.6214163614163614
Accuracy en entrenamiento: 0.6600441501103753
Matriz de Confusión:
[[ 32  42   0   0]
 [ 18 314  42  12]
 [  1  97  76  58]
 [  0  13  25 176]]
```

Los resultados no parecen mejorar con respecto a intentos anteriores. Probamos a aplicar el clasificador del intento 2 pero sin regularización.

[231]:
```
X_train = data_train_num_imp
y_train = data_train['RATE']

X_test = data_test_num_imp
```

[234]:
```
from bagging import Bagging

import warnings
warnings.filterwarnings('ignore')
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix, accuracy_score
```

```
# Modelo de regresión logística
model_logreg = LogisticRegression(penalty=None, class_weight=None, max_iter=300)

# Entrenamiento del modelo con Bagging
bag = Bagging([model_logreg], 50, random_state=42)

# Validación cruzada estratificada 10-fold
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
accs = []
for train_index, test_index in kf.split(X_train, y_train):
    X_train_fold = X_train.iloc[train_index]
    X_test_fold = X_train.iloc[test_index]
    y_train_fold = y_train[train_index]
    y_test_fold = y_train[test_index]

    # Entrenamos y evaluamos cada fold
    bag.fit(X_train_fold, y_train_fold)
    accs.append(accuracy_score(y_test_fold, bag.predict(X_test_fold)))

print("Accuracy en validación cruzada:", np.mean(accs))

# Predicciones en entrenamiento
bag.fit(X_train, y_train)
y_train_pred = bag.predict(X_train)
print("Accuracy en entrenamiento:", accuracy_score(y_train, y_train_pred))

# Matriz de confusión
conf_matrix = confusion_matrix(y_train, y_train_pred)
print("Matriz de Confusión:")
print(conf_matrix)
```

```
Accuracy en validación cruzada: 0.6268009768009767
Accuracy en entrenamiento: 0.6920529801324503
Matriz de Confusión:
[[ 42  32   0   0]
 [ 16 312  45  13]
 [  1  75 102  54]
 [  0  10  33 171]]
```

[233]:
```
y_test_pred = bag.predict(X_test)

# Número de instancias etiquetadas en cada clase
print(pd.Series(y_test_pred).value_counts())
```

```
0
B    188
D    109
C     69
```

```
A     23
Name: count, dtype: int64
```

[236]:
```python
# Resultados con el formato de entrega para Kaggle
results5 = pd.DataFrame({'ID': data_test['ID'], 'RATE': y_test_pred})
print(results5.head())
print(results5.shape)

results5.to_csv('results5.csv', index=False)
```

```
    ID RATE
0  656    B
1  174    B
2  129    D
3  674    D
4  197    B
(389, 2)
```