

BAF_xgboost_multiclass28102019_1237am

December 12, 2019

1 Predicción de nivel de penetracion de Banda Ancha fija (fibra óptica o cable coaxial) en municipios

El problema a evaluar es el nivel de penetración de BAF basada en fibra óptica o cable coaxial en los municipios (ver nota explicativa de la definición del nivel de penetración, que tomó como referencia los datos de países de la OCDE a finales de 2018).

Nota: Se empleó un modelo de regresión logística, XGBoost y Random Forest

1.0.1 Preámbulo de paquetes a utilizarse

```
[1]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold

import xgboost as xgb
import shap
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

1.0.2 Carga de datos de penetracion BAF y municipios

```
[2]: # Cargamos los datos en crudo y creamos una copia para trabajar
raw_data = pd.read_csv ('BAF_06209_P2.csv')
data = raw_data

# Elimina columnas de municipio y clases de penetracion (solo estudiaremos si
→hay
# o no penetracion de F.O y Cable Coax.)
del data['K_ENTIDAD_MUNICIPIO']
del data['IS_PEN_BAF_HABS_COAXFO'] #
del data['ANALF'] #

# Renombramos una columna para que no genere problemas con Scikit Learn
```

```
data.rename(columns={"PL<5000": "PL5000"}, inplace = True)

# Renombramos una columnas para facilitar manipulacion
data.rename(columns={"ANOS_PROMEDIO_DE_ESCOLARIDAD": "AVG_ESCOLARIDAD"},
            inplace = True )
data.rename(columns={"CLASS_PEN_BAF_HABS_COAXFO": "LEVEL_PEN"}, inplace = True)

# Sustituimos valores de la columnas NUM_OPS (1= Hay mas de dos operadores, 0 =
            en otro caso)
data['NUM_OPS'] = np.where(data['NUM_OPS']>1,1,0)

# Descartamos los municipios que no poseen indicadores de ingreso anual bruto
            per capita
data.dropna(inplace = True)
```

1.0.3 Resumen de las variables en estudio

[3]: data.describe()

```
[3]:
```

	HOGARES	POBLACION	SUPERFICIE	DENS_HOGS	\
count	2446.000000	2.446000e+03	2446.000000	2446.000000	
mean	12991.372036	4.860599e+04	791.321885	7916.018110	
std	38143.144359	1.389142e+05	2104.590464	35001.243252	
min	31.000000	8.700000e+01	2.210000	4.995331	
25%	1111.750000	4.253000e+03	85.765000	521.867640	
50%	3471.500000	1.340400e+04	233.485000	1387.721595	
75%	8877.500000	3.439950e+04	654.895000	3462.189955	
max	495665.000000	1.827868e+06	53138.790000	598127.340824	

	DENS_HABS	SPRIM	AVG_ESCOLARIDAD	OVSAAE	OVSEE	\
count	2446.000000	2446.000000	2446.000000	2446.000000	2446.000000	
mean	7916.018110	29.213050	6.423426	8.651132	2.169366	
std	35001.243252	11.862555	1.767869	11.487796	3.410109	
min	4.995331	2.490000	1.460000	0.000000	0.000000	
25%	521.867640	20.500000	5.190000	1.492500	0.490000	
50%	1387.721595	29.405000	6.280000	4.060000	1.175000	
75%	3462.189955	37.345000	7.520000	10.865000	2.560000	
max	598127.340824	71.240000	13.830000	98.880000	57.960000	

	PL5000	P02SM	INGRESOPC_ANUAL	DISP_INTERNET	DISP_TV_PAGA	\
count	2446.000000	2446.000000	2446.000000	2446.000000	2446.000000	
mean	71.898684	55.381063	1935.145724	12.054015	31.158202	
std	34.685724	16.985484	1020.473455	12.770534	18.829657	
min	0.000000	8.250000	185.290000	0.000000	0.000000	
25%	42.687500	42.930000	1198.730000	2.100553	15.313693	
50%	100.000000	57.020000	1789.905000	7.954121	29.018092	
75%	100.000000	68.470000	2447.497500	17.788759	45.173174	

max	100.000000	94.120000	9748.530000	81.882586	85.097192
-----	------------	-----------	-------------	-----------	-----------

	DISP_TEL_CELULAR	DISP_TEL_FIJO	NUM_OPS	LEVEL_PEN
count	2446.000000	2446.000000	2446.000000	2446.000000
mean	57.439859	20.204734	0.226492	0.401881
std	25.072398	14.044259	0.418647	0.661519
min	0.000000	0.000000	0.000000	0.000000
25%	42.198730	9.047367	0.000000	0.000000
50%	64.877177	17.645177	0.000000	0.000000
75%	76.445478	29.454433	0.000000	1.000000
max	95.002027	84.047612	1.000000	4.000000

1.0.4 Variables de la base

```
[4]: list(data.columns.values)
```

```
[4]: ['HOGARES',
      'POBLACION',
      'SUPERFICIE',
      'DENS_HOGS',
      'DENS_HABS',
      'SPRIM',
      'AVG_ESCOLARIDAD',
      'OVSAE',
      'OVSEE',
      'PL5000',
      'PO2SM',
      'INGRESOPC_ANUAL',
      'DISP_INTERNET',
      'DISP_TV_PAGA',
      'DISP_TEL_CELULAR',
      'DISP_TEL_FIJO',
      'NUM_OPS',
      'LEVEL_PEN']
```

1.0.5 Dimensiones de la base

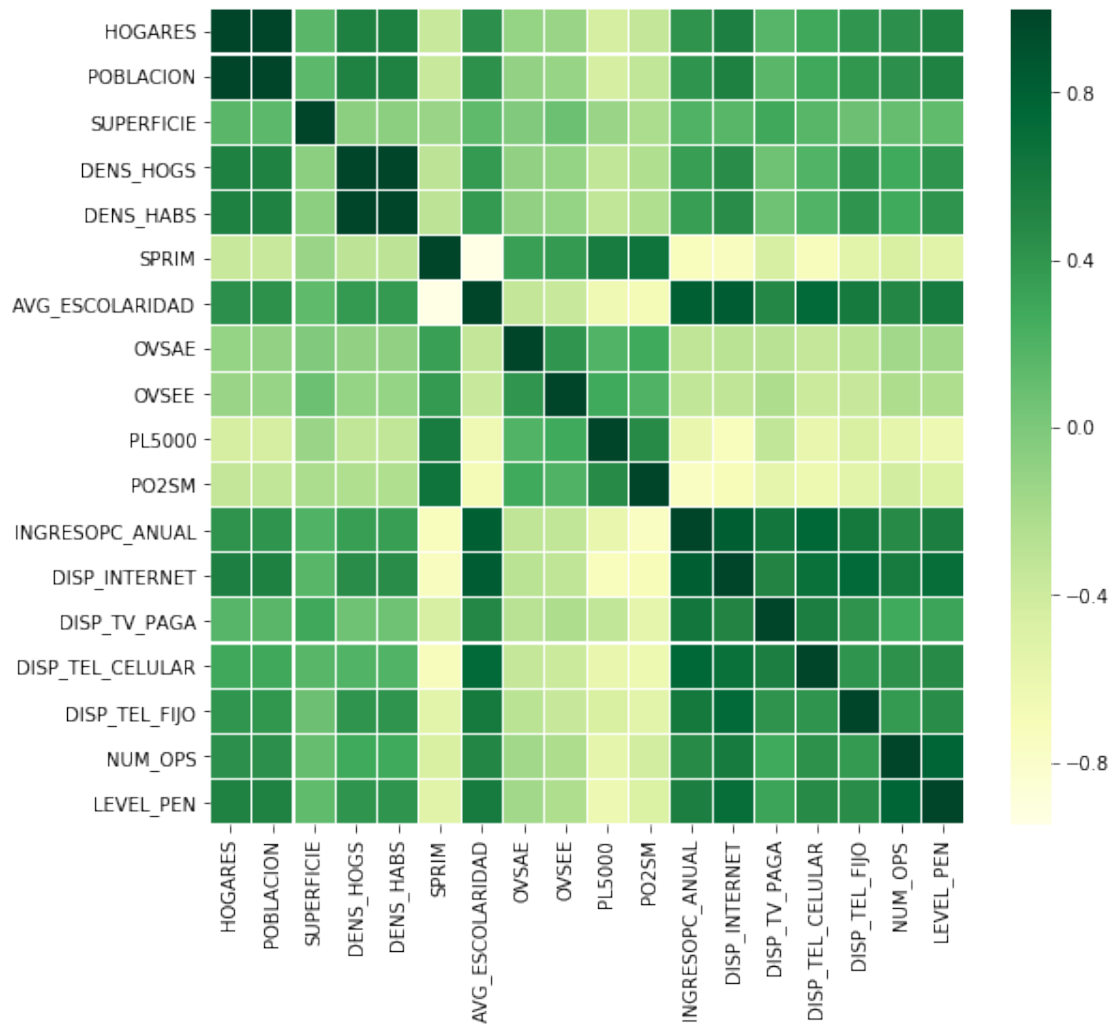
```
[5]: m, n = data.shape
```

1.0.6 Matrices de correlación

```
[6]: corrmatrix = data.corr()

f, ax = plt.subplots(figsize=(9, 8))
sns.heatmap(corrmatrix, ax=ax, cmap="YlGn", linewidths=0.1)
```

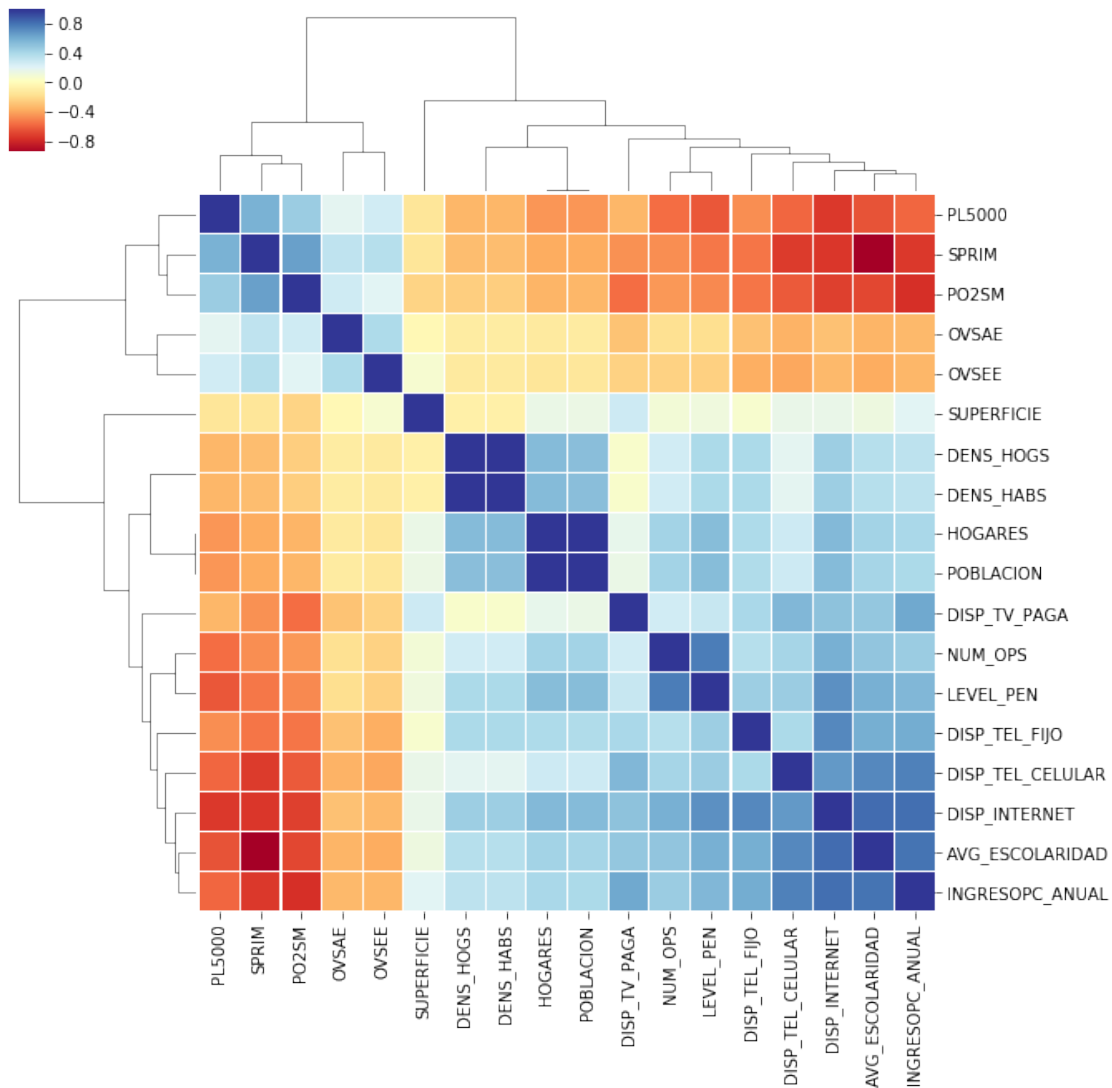
```
[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7f14294e1668>
```



```
[7]: #corrmat = data.corr()
cg = sns.clustermap(corrmat, cmap = "RdYlBu", linewidths = 0.1);
plt.setp(cg.ax_heatmap.yaxis.get_majorticklabels(), rotation = 0)
```

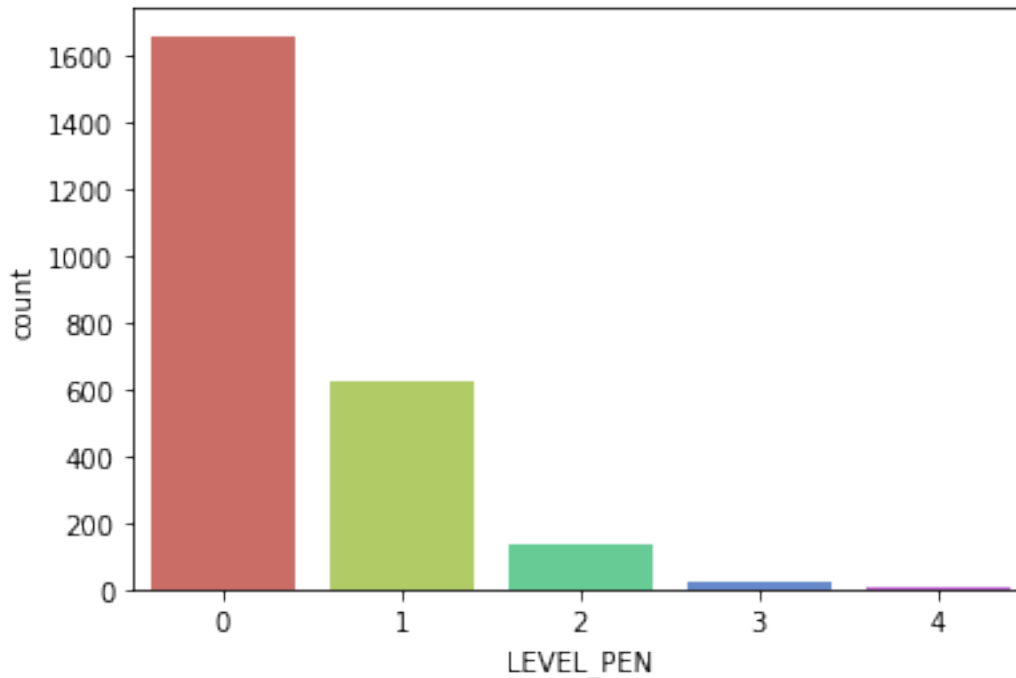
```
cg
```

```
[7]: <seaborn.matrix.ClusterGrid at 0x7f1429566860>
```



```
[8]: sns.countplot(x='LEVEL_PEN',data=data,palette='hls')
data.groupby(['LEVEL_PEN']).size()
```

```
[8]: LEVEL_PEN
0    1660
1     625
2     133
3       20
4        8
dtype: int64
```



1.0.7 Preparamos los datos para entrenamiento y prueba

```
[9]: from sklearn.model_selection import train_test_split
seed = 712 # Seed para poder replicar
X_train, X_test, y_train, y_test = train_test_split(data.iloc[:, :n-1], data.
    ↳iloc[:, n-1], random_state=seed)

xg_train = xgb.DMatrix(X_train, label=y_train)
xg_test = xgb.DMatrix(X_test, label=y_test)
```

/home/cesar/.local/lib/python3.7/site-packages/xgboost/core.py:587:

FutureWarning: Series.base is deprecated and will be removed in a future version
if getattr(data, 'base', None) is not None and \

1.0.8 Modelo con regresion logística 1 (All vs All)

```
[10]: from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
```

```
[11]: model_lr1 = LogisticRegression(random_state=seed, solver='lbfgs',
    ↳max_iter=1000000, multi_class = "multinomial").fit(preprocessing.
    ↳scale(X_train), y_train)
```

```
[12]: y_model_lr1_predicted = model_lr1.predict(preprocessing.scale(X_test))
```

```
[13]: ## Accuracy
from sklearn.metrics import accuracy_score
print("Accuracy: {}".format(accuracy_score(y_test, y_model_lr1_predicted)))

## Métricas clásicas de Clasificadores
from sklearn import metrics
print(metrics.classification_report(y_model_lr1_predicted, y_test))

## Matriz de confusión
from sklearn.metrics import confusion_matrix

mat_lr = confusion_matrix(y_test, y_model_lr1_predicted)

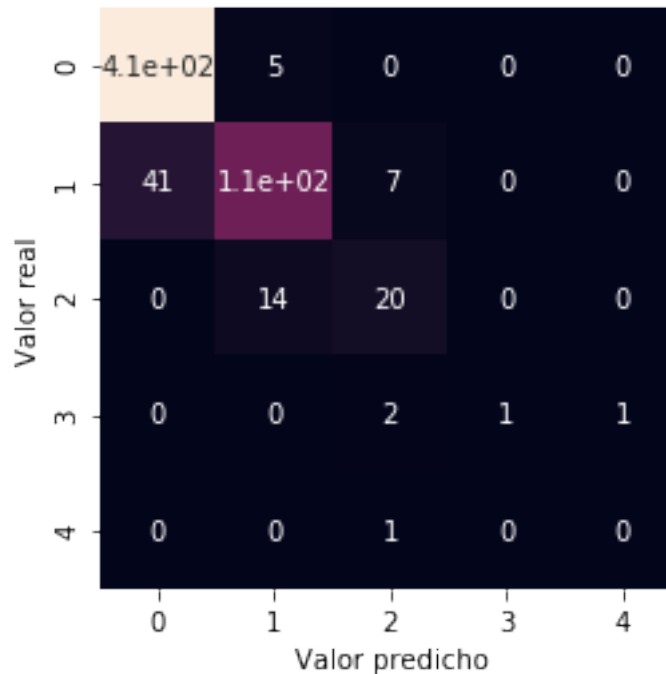
plt.close()

sns.heatmap(mat_lr, square = True, annot = True, cbar = False)
plt.xlabel('Valor predicho')
plt.ylabel('Valor real')
```

Accuracy: 0.8839869281045751

	precision	recall	f1-score	support
0	0.99	0.91	0.95	447
1	0.70	0.86	0.77	133
2	0.59	0.67	0.62	30
3	0.25	1.00	0.40	1
4	0.00	0.00	0.00	1
accuracy			0.88	612
macro avg	0.51	0.69	0.55	612
weighted avg	0.90	0.88	0.89	612

[13]: Text(91.68, 0.5, 'Valor real')



1.0.9 Modelo con regresion logística 2 (One vs All)

```
[14]: model_lr2 = LogisticRegression(random_state=seed,solver='sag',
    ↳max_iter=1000000, multi_class = "ovr").fit(preprocessing.scale(X_train),
    ↳y_train)

[15]: y_model_lr2_predicted = model_lr2.predict(preprocessing.scale(X_test))

[16]: ## Accuracy
from sklearn.metrics import accuracy_score
print("Accuracy: {}".format(accuracy_score(y_test, y_model_lr1_predicted)))

## Métricas clásicas de Clasificadores
from sklearn import metrics
print(metrics.classification_report(y_model_lr1_predicted, y_test))

## Matriz de confusión
from sklearn.metrics import confusion_matrix

mat_lr = confusion_matrix(y_test, y_model_lr1_predicted)

plt.close()

sns.heatmap(mat_lr, square = True, annot = True, cbar = False)
plt.xlabel('Valor predicho')
```

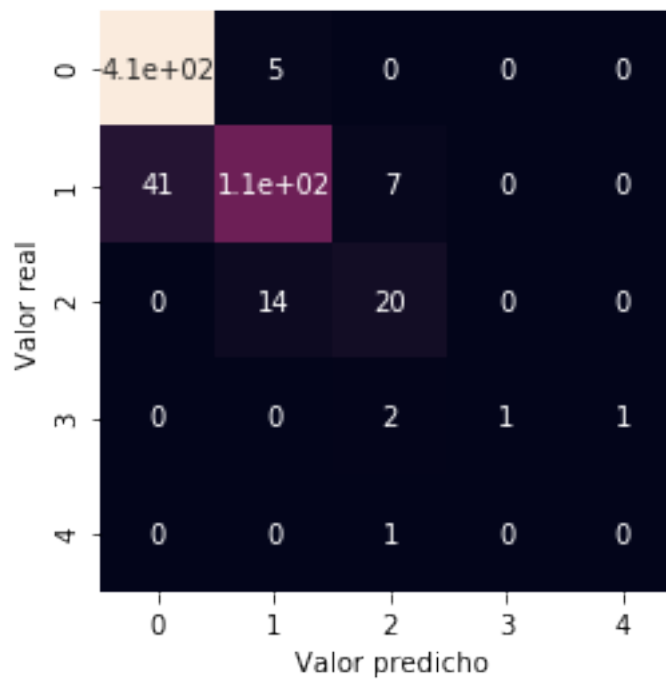


```
plt.ylabel('Valor real')
```

Accuracy: 0.8839869281045751

	precision	recall	f1-score	support
0	0.99	0.91	0.95	447
1	0.70	0.86	0.77	133
2	0.59	0.67	0.62	30
3	0.25	1.00	0.40	1
4	0.00	0.00	0.00	1
accuracy			0.88	612
macro avg	0.51	0.69	0.55	612
weighted avg	0.90	0.88	0.89	612

[16]: Text(91.68, 0.5, 'Valor real')



1.0.10 Modelo con XGBoost

```
[17]: from sklearn.model_selection import GridSearchCV

# Instanciamos el clasificador
classifier_xgb = xgb.XGBClassifier()
```

```

# Hiper-parametros para hacer el grid search
hyper_param_grid = {
    'objective': ['multi:softmax'],
    'eta': [0.01, 0.05, 0.1], # Tasa de aprendizaje
    'gamma': [0.5],
    'max_depth': [6, 7, 8],
    'subsample': [0.8],
    'silent': [1],
    'nthread': [10],
    'num_class': [5],
    'subsample': [0.5],
    'colsample_bytree': [1.0],
    'n_estimators': [10, 100, 100], #number of trees, change it to 1000 for
    better results
    'seed': [27]
}

model_gsearch_xgb = GridSearchCV(classifier_xgb, hyper_param_grid, cv = 6,
    verbose = 2)

model_gsearch_xgb.fit(X_train, y_train, eval_metric='auc')

```

Fitting 6 folds for each of 27 candidates, totalling 162 fits

[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=6, n_estimators=10, nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1, subsample=0.5

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=6, n_estimators=10, nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1, subsample=0.5, total= 0.4s

[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=6, n_estimators=10, nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1, subsample=0.5

[Parallel(n_jobs=1)]: Done 1 out of 1 | elapsed: 0.4s remaining: 0.0s

[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=6, n_estimators=10, nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1, subsample=0.5, total= 0.3s

[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=6, n_estimators=10, nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1, subsample=0.5

[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=6, n_estimators=10, nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1, subsample=0.5, total= 0.3s

[illegible]

[illegible]

```
[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=7, n_estimators=100,  
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,  
subsample=0.5  
[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=7, n_estimators=100,  
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,  
subsample=0.5, total=    1.7s  
[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=7, n_estimators=100,  
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,  
subsample=0.5  
[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=7, n_estimators=100,  
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,  
subsample=0.5, total=    1.7s  
[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=7, n_estimators=100,  
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,  
subsample=0.5  
[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=7, n_estimators=100,  
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,  
subsample=0.5, total=    1.7s  
[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=7, n_estimators=100,  
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,  
subsample=0.5  
[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=7, n_estimators=100,  
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,  
subsample=0.5, total=    1.8s  
[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=7, n_estimators=100,  
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,  
subsample=0.5  
[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=7, n_estimators=100,  
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,  
subsample=0.5, total=    1.8s  
[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=7, n_estimators=100,  
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,  
subsample=0.5  
[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=7, n_estimators=100,  
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,  
subsample=0.5, total=    1.7s  
[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=7, n_estimators=100,  
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,  
subsample=0.5  
[CV] colsample_bytree=1.0, eta=0.01, gamma=0.5, max_depth=7, n_estimators=100,  
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,  
subsample=0.5, total=    1.7s
```


[illegible]

[illegible]

[illegible]

[illegible]

```
[CV] colsample_bytree=1.0, eta=0.05, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.05, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 1.8s
[CV] colsample_bytree=1.0, eta=0.05, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.05, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 1.8s
[CV] colsample_bytree=1.0, eta=0.05, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.05, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 1.9s
[CV] colsample_bytree=1.0, eta=0.05, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.05, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 2.0s
[CV] colsample_bytree=1.0, eta=0.05, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.05, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 2.4s
[CV] colsample_bytree=1.0, eta=0.05, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.05, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 2.0s
[CV] colsample_bytree=1.0, eta=0.05, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.05, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 1.9s
```

```

[CV] colsample_bytree=1.0, eta=0.05, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.05, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 2.0s
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=6, n_estimators=10,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=6, n_estimators=10,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 0.2s
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=6, n_estimators=10,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=6, n_estimators=10,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 0.3s
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=6, n_estimators=10,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=6, n_estimators=10,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 0.3s
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=6, n_estimators=10,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=6, n_estimators=10,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 0.3s
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=6, n_estimators=10,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=6, n_estimators=10,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 0.4s
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=6, n_estimators=10,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=6, n_estimators=10,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 0.2s
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=6, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=6, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 1.7s

```


[illegible]

```

[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 1.9s
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 1.9s
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 1.9s
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 1.9s
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 1.9s
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5
[CV] colsample_bytree=1.0, eta=0.1, gamma=0.5, max_depth=8, n_estimators=100,
nthread=10, num_class=5, objective=multi:softmax, seed=27, silent=1,
subsample=0.5, total= 1.9s
[Parallel(n_jobs=1)]: Done 162 out of 162 | elapsed: 3.6min finished

```

```

[17]: GridSearchCV(cv=6, error_score='raise-deprecating',
        estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                colsample_bylevel=1, colsample_bynode=1,

```

```

        colsample_bytree=1, gamma=0,
        learning_rate=0.1, max_delta_step=0,
        max_depth=3, min_child_weight=1,
        missing=None, n_estimators=100, n_jobs=1,
        nthread=None, objective='binary:logistic',
        random_state=0, reg_alpha=0, reg_l...
        subsample=1, verbosity=1),
    iid='warn', n_jobs=None,
    param_grid={'colsample_bytree': [1.0], 'eta': [0.01, 0.05, 0.1],
                'gamma': [0.5], 'max_depth': [6, 7, 8],
                'n_estimators': [10, 100, 100], 'nthread': [10],
                'num_class': [5], 'objective': ['multi:softmax'],
                'seed': [27], 'silent': [1], 'subsample': [0.5]},
    pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
    scoring=None, verbose=2)

```

```

[18]: #model_gsearch_xgb.grid_scores_
      model_gsearch_xgb.best_params_

```

```

[18]: {'colsample_bytree': 1.0,
      'eta': 0.01,
      'gamma': 0.5,
      'max_depth': 7,
      'n_estimators': 10,
      'nthread': 10,
      'num_class': 5,
      'objective': 'multi:softmax',
      'seed': 27,
      'silent': 1,
      'subsample': 0.5}

```

```

[19]: model_gsearch_xgb.best_score_

```

```

[19]: 0.8833151581243184

```

```

[]:

```

```

[]:

```

```

[20]: y_model_xgb = model_gsearch_xgb.predict(X_test)
      ## Accuracy
      from sklearn.metrics import accuracy_score
      print("Accuracy: {}".format(accuracy_score(y_test, y_model_xgb)))

      ## Métricas clásicas de Clasificadores
      from sklearn import metrics
      print(metrics.classification_report(y_model_xgb, y_test))

      ## Matriz de confusión
      from sklearn.metrics import confusion_matrix

```

```

mat = confusion_matrix(y_test, y_model_xgb)

plt.close()

sns.heatmap(mat, square = True, annot = True, cbar = False)
plt.xlabel('Valor predicho')
plt.ylabel('Valor real')

```

Accuracy: 0.8790849673202614

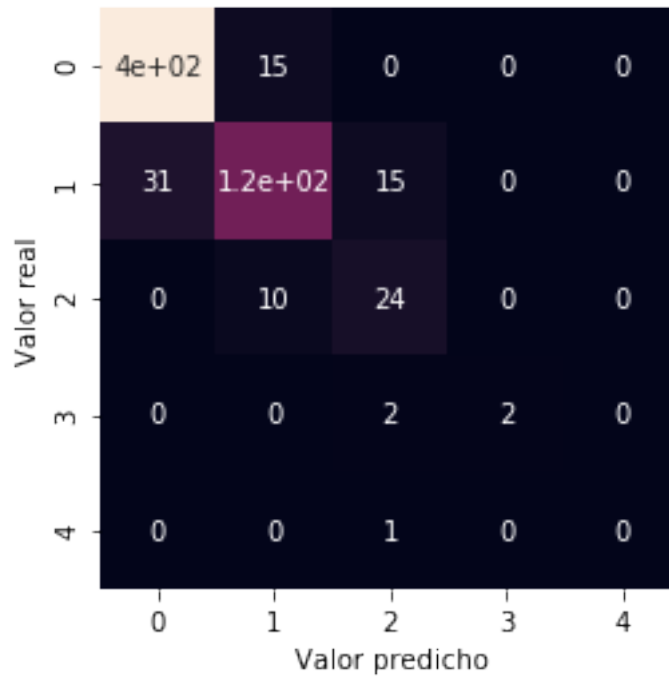
	precision	recall	f1-score	support
0	0.96	0.93	0.95	427
1	0.72	0.82	0.77	141
2	0.71	0.57	0.63	42
3	0.50	1.00	0.67	2
4	0.00	0.00	0.00	0
accuracy			0.88	612
macro avg	0.58	0.66	0.60	612
weighted avg	0.89	0.88	0.88	612

```

/home/cesar/anaconda3/lib/python3.7/site-
packages/sklearn/metrics/classification.py:1439: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
'recall', 'true', average, warn_for)

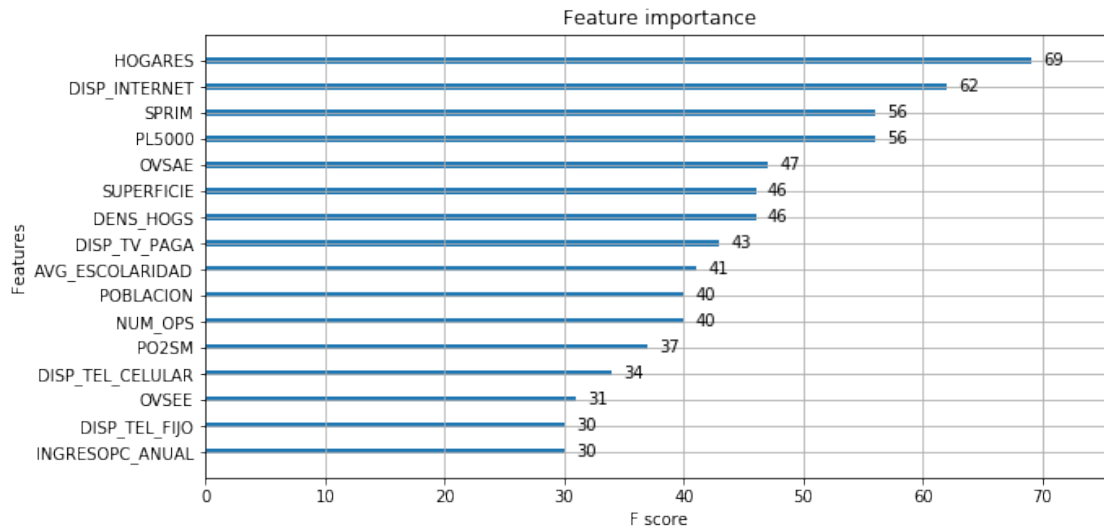
```

[20]: Text(91.68, 0.5, 'Valor real')



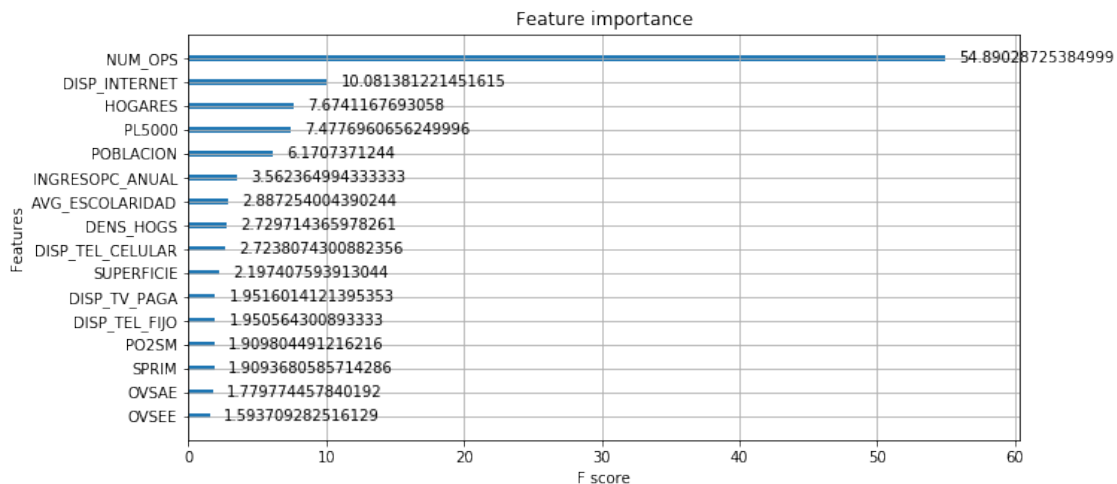
```
[21]: # How the importance is calculated: either "weight", "gain", or "cover"
#      "weight" is the number of times a feature appears in a tree
#      "gain" is the average "gain" of splits which use the feature
#      "cover" is the average coverage of splits which use the feature
#      where coverage is defined as the number of samples affected by the split

fig, ax = plt.subplots(figsize=(10,5))
xgb.plot_importance(model_gsearch_xgb.best_estimator_, ax=ax,
                    importance_type='weight')
plt.show()
```



[22]: *# How the importance is calculated: either "weight", "gain", or "cover"*
"weight" is the number of times a feature appears in a tree
"gain" is the average "gain" of splits which use the feature
"cover" is the average coverage of splits which use the feature
where coverage is defined as the number of samples affected by the split

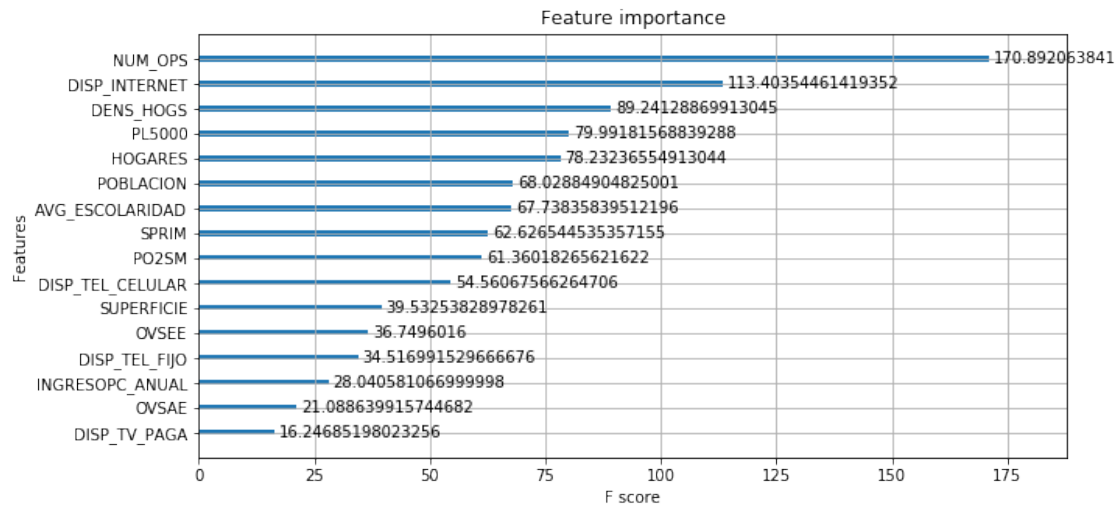
```
fig, ax = plt.subplots(figsize=(10,5))
xgb.plot_importance(model_gsearch_xgb.best_estimator_, ax=ax,
    ↪importance_type='gain')
plt.show()
```



[23]: *# How the importance is calculated: either "weight", "gain", or "cover"*
"weight" is the number of times a feature appears in a tree

```
# "gain" is the average "gain" of splits which use the feature
# "cover" is the average coverage of splits which use the feature
# where coverage is defined as the number of samples affected by the split

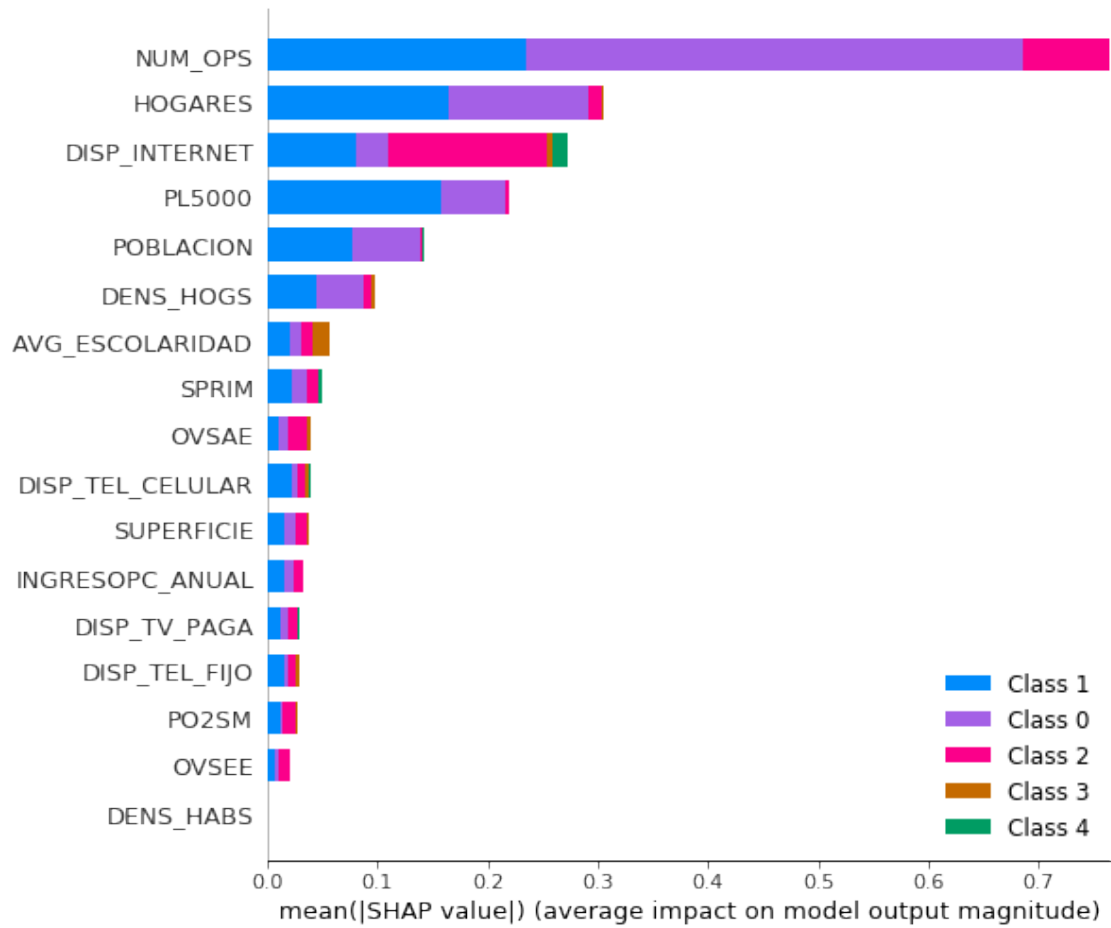
fig, ax = plt.subplots(figsize=(10,5))
xgb.plot_importance(model_gsearch_xgb.best_estimator_, ax=ax,
    →importance_type='cover')
plt.show()
```



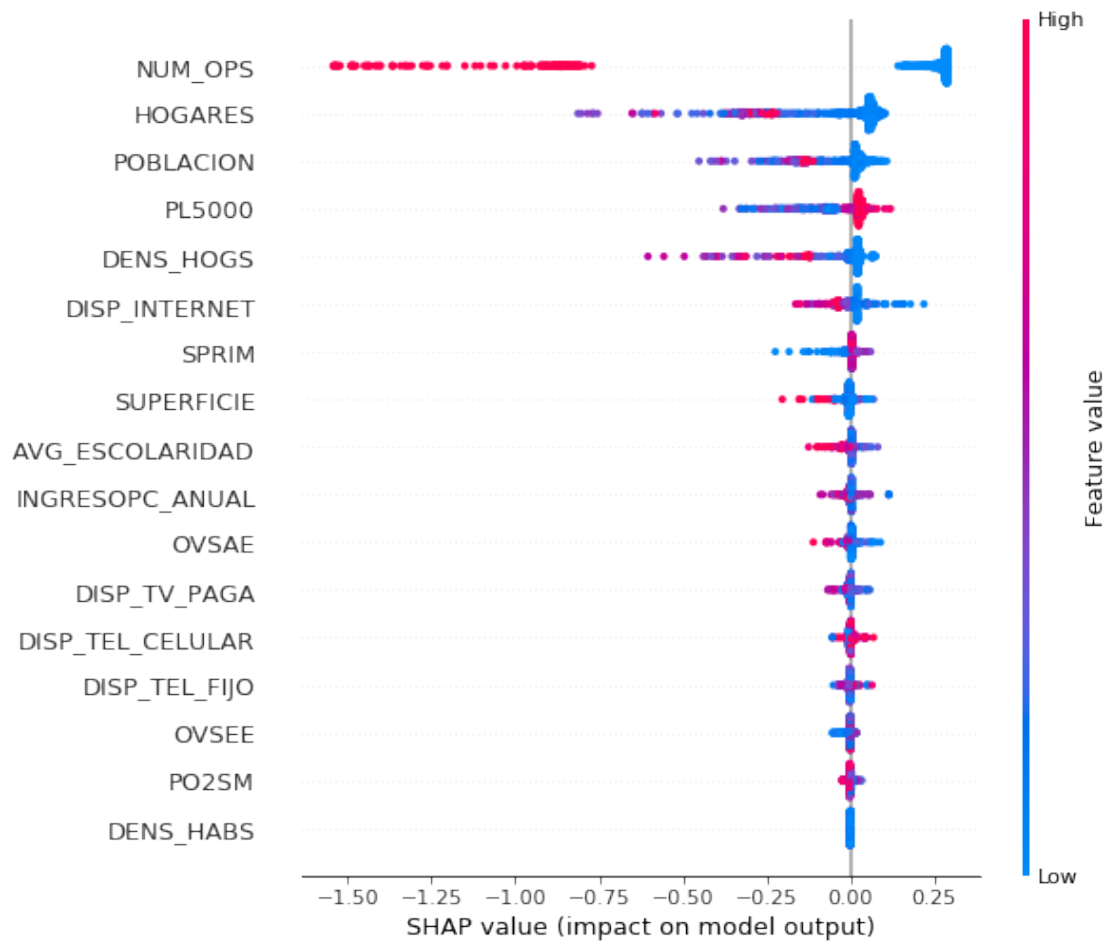
```
[24]: # Create object that can calculate shap values
explainer = shap.TreeExplainer(model_gsearch_xgb.best_estimator_)

# Calculate shap_values for all of val_X rather than a single row, to have more
→data for plot.
shap_values = explainer.shap_values(X_test)

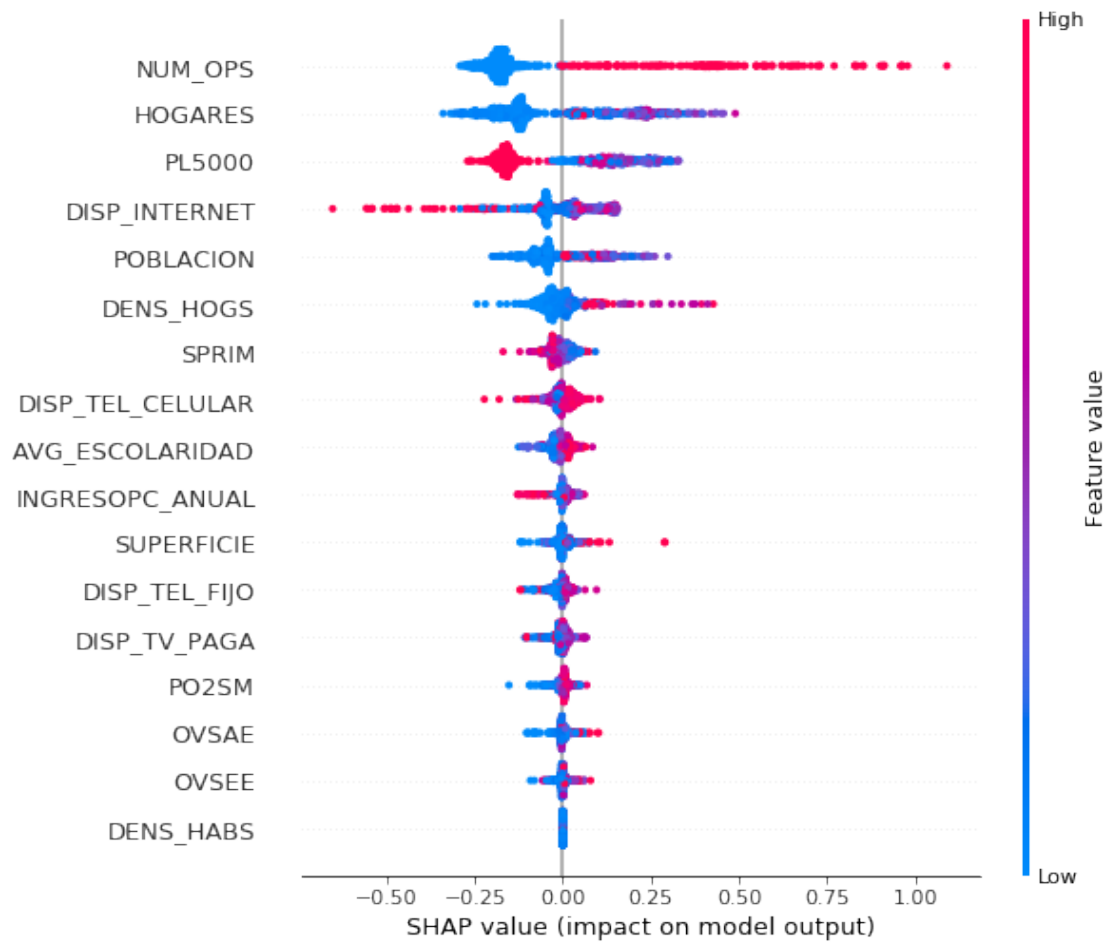
[25]: shap.summary_plot(shap_values, X_test) # Plot de los que no tienen conectividad
→de BAF (f.o. + coaxial)
```



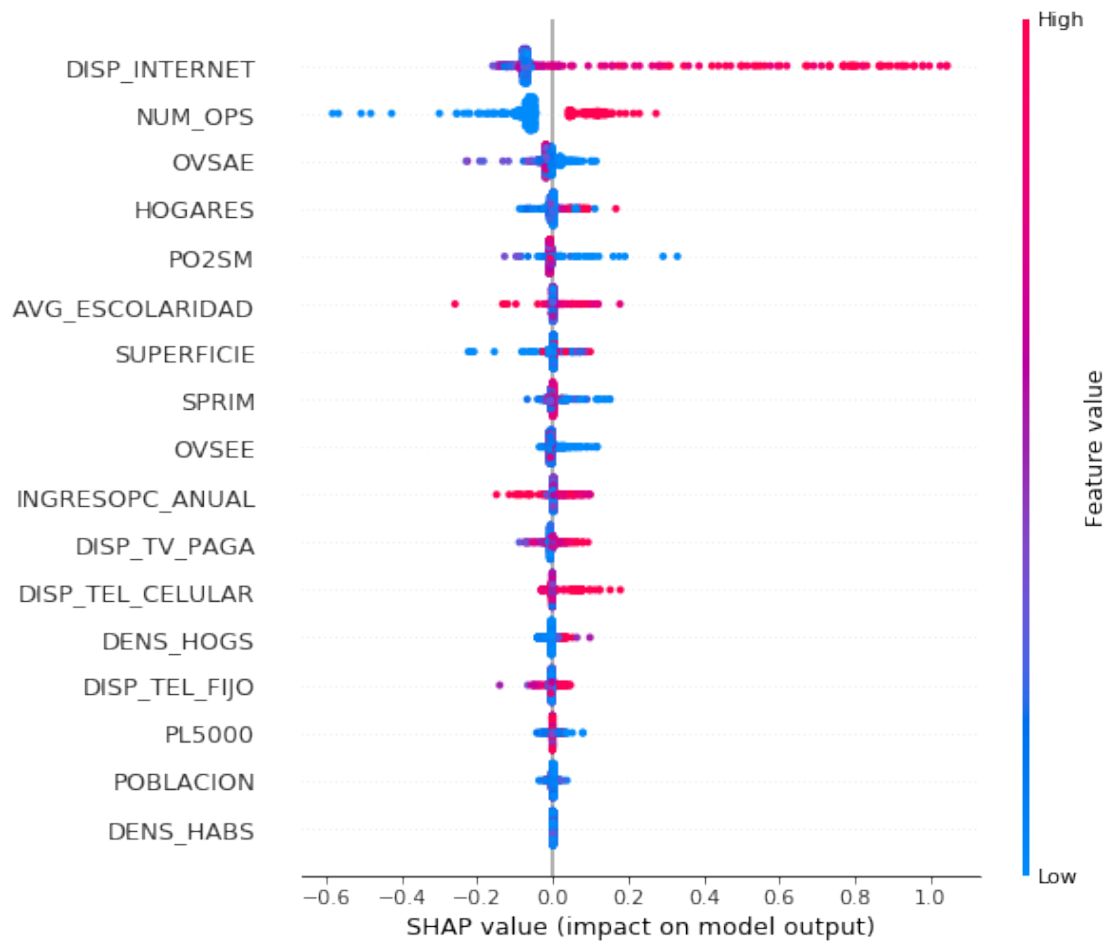
[26]: *# Make plot. Index of [0] is explained in text below.*
`shap.summary_plot(shap_values[0], X_test)` *# Plot de los que no tienen*
→conectividad de BAF (f.o. + coaxial)



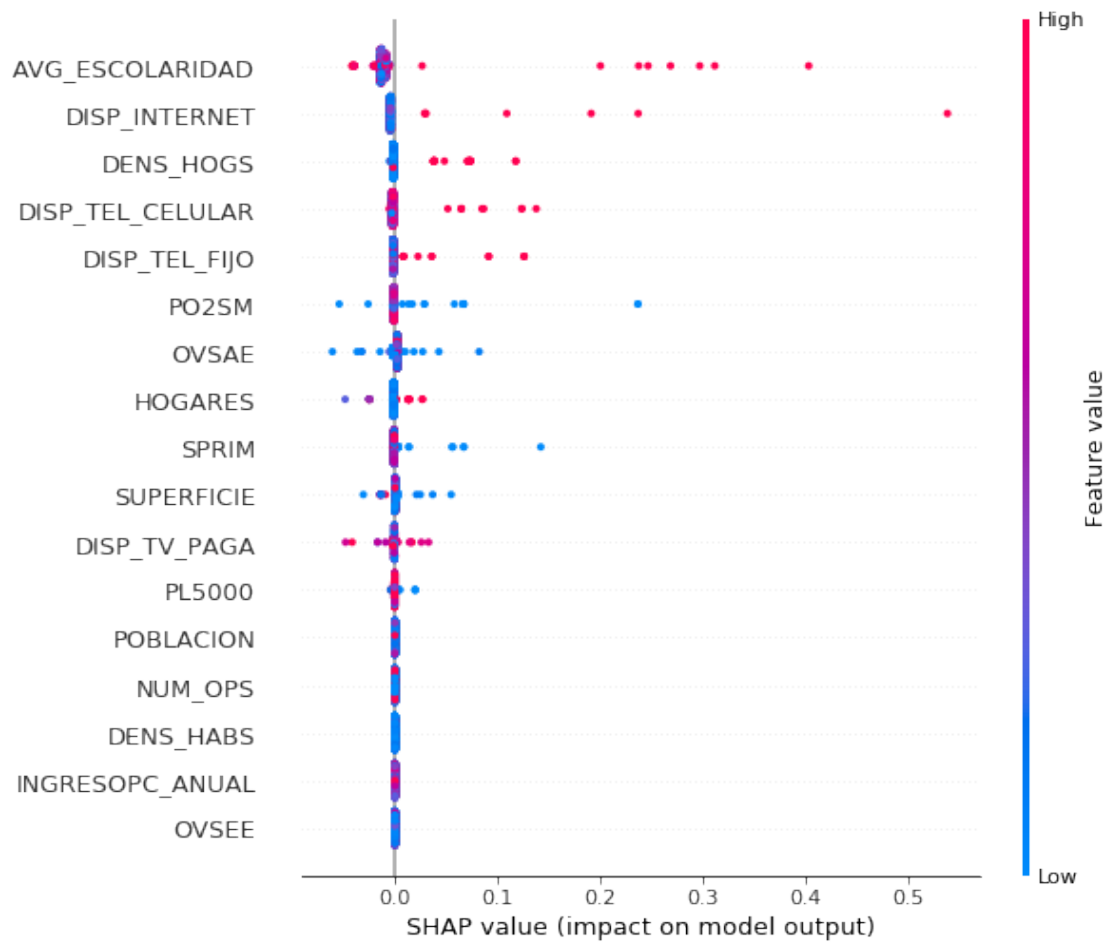
```
[27]: shap.summary_plot(shap_values[1], X_test) # Plot de los que si tienen
      ↪ conectividad de BAF (f.o. + coaxial)
```



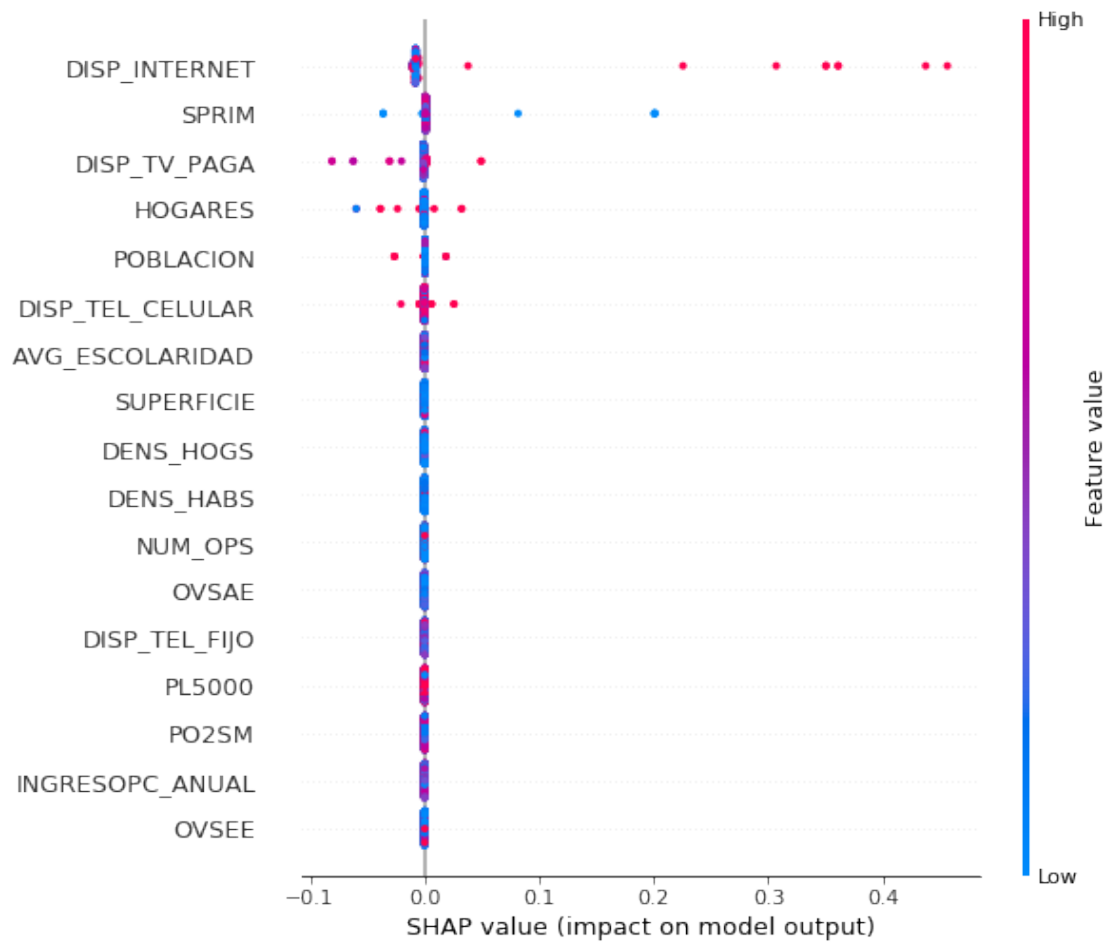
```
[28]: shap.summary_plot(shap_values[2], X_test) # Plot de los que si tienen
      ↪ conectividad de BAF (f.o. + coaxial)
```



```
[29]: shap.summary_plot(shap_values[3], X_test) # Plot de los que si tienen
      ↪ conectividad de BAF (f.o. + coaxial)
```



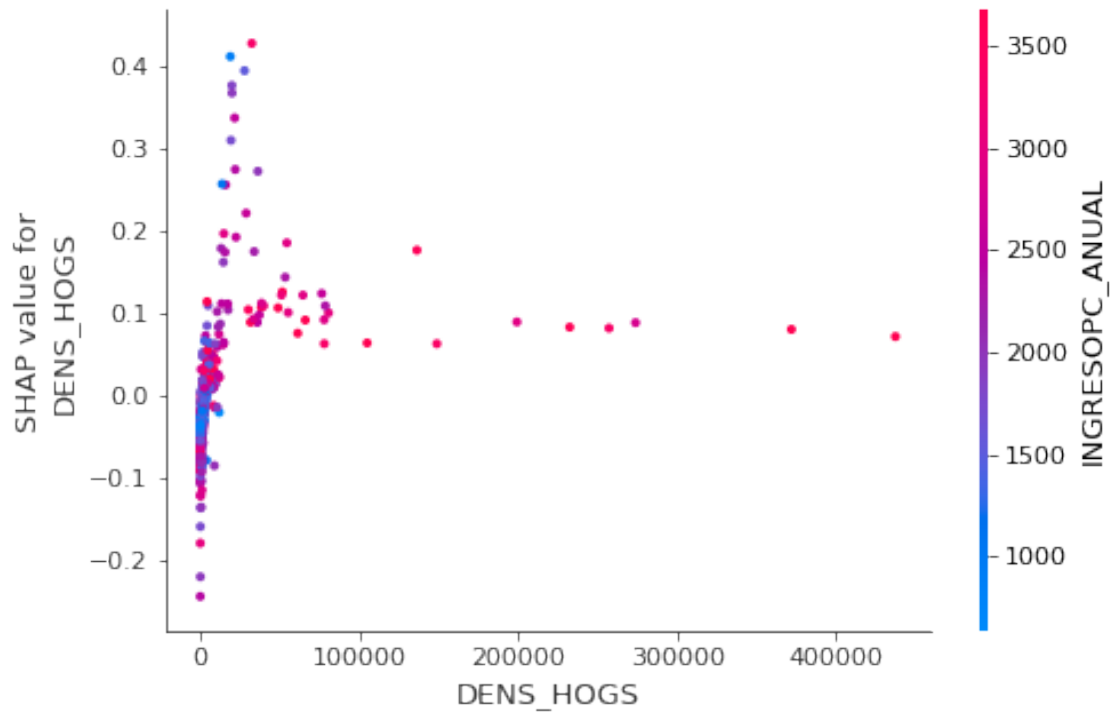
```
[30]: shap.summary_plot(shap_values[4], X_test) # Plot de los que si tienen
      ↪ conectividad de BAF (f.o. + coaxial)
```

[]:

[32]: # <https://www.kaggle.com/dansbecker/advanced-uses-of-shap-values>

```
shap.dependence_plot('DENS_HOGS', shap_values[1], X_test,
    ↳interaction_index="INGRESOPC_ANUAL")
```



1.0.11 Random forests

```
[33]: from sklearn.model_selection import GridSearchCV

rf_classifier = RandomForestClassifier()
hyper_param_grid = {'n_estimators': [1,10,100],#[1,10,100,1000,10000],
                    'max_depth': [5,10,20,50,100],#[5,10,20,50,100],
                    'max_features': ['sqrt','log2'],
                    'min_samples_split': [2,5,10],
                    'criterion':['entropy'],
                    #'class_weight':['balanced'],
                    'warm_start' :['TRUE'],
                    'random_state':[712]
                    },

model_RF_gsearch = GridSearchCV(rf_classifier, hyper_param_grid, cv = 6,
    verbose = 3)

model_RF_gsearch.fit(X_train, y_train)

y_model_forest = model_RF_gsearch.predict(X_test)
```

```

Fitting 6 folds for each of 90 candidates, totalling 540 fits
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=1, random_state=712, warm_start=TRUE
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=1, random_state=712, warm_start=TRUE, score=0.832, total= 0.0s
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=1, random_state=712, warm_start=TRUE
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=1, random_state=712, warm_start=TRUE, score=0.856, total= 0.0s
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=1, random_state=712, warm_start=TRUE
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=1, random_state=712, warm_start=TRUE, score=0.807, total= 0.0s
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=1, random_state=712, warm_start=TRUE
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=1, random_state=712, warm_start=TRUE, score=0.852, total= 0.0s
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=1, random_state=712, warm_start=TRUE
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=1, random_state=712, warm_start=TRUE, score=0.852, total= 0.0s
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=1, random_state=712, warm_start=TRUE
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=1, random_state=712, warm_start=TRUE, score=0.829, total= 0.0s
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=10, random_state=712, warm_start=TRUE
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=10, random_state=712, warm_start=TRUE, score=0.896, total= 0.1s
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=10, random_state=712, warm_start=TRUE
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1 out of 1 | elapsed: 0.0s remaining: 0.0s
[Parallel(n_jobs=1)]: Done 2 out of 2 | elapsed: 0.0s remaining: 0.0s

[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=10, random_state=712, warm_start=TRUE, score=0.915, total= 0.1s
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=10, random_state=712, warm_start=TRUE
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=10, random_state=712, warm_start=TRUE, score=0.837, total= 0.1s
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=10, random_state=712, warm_start=TRUE
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=10, random_state=712, warm_start=TRUE, score=0.905, total= 0.1s
[CV] criterion=entropy, max_depth=5, max_features=sqrt, min_samples_split=2,
n_estimators=10, random_state=712, warm_start=TRUE

```

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
[CV] criterion=entropy, max_depth=100, max_features=log2, min_samples_split=10,
n_estimators=100, random_state=712, warm_start=TRUE, score=0.855, total= 0.7s
[CV] criterion=entropy, max_depth=100, max_features=log2, min_samples_split=10,
n_estimators=100, random_state=712, warm_start=TRUE
[CV] criterion=entropy, max_depth=100, max_features=log2, min_samples_split=10,
n_estimators=100, random_state=712, warm_start=TRUE, score=0.875, total= 0.7s

[Parallel(n_jobs=1)]: Done 540 out of 540 | elapsed: 2.4min finished
```

```
[34]: model_RF_gsearch.best_params_
```

```
[34]: {'criterion': 'entropy',
      'max_depth': 10,
      'max_features': 'sqrt',
      'min_samples_split': 2,
      'n_estimators': 100,
      'random_state': 712,
      'warm_start': 'TRUE'}
```

```
[35]: model_RF_gsearch.best_score_
```

```
[35]: 0.8854961832061069
```

```
[36]: ## Accuracy
from sklearn.metrics import accuracy_score
print("Accuracy: {}".format(accuracy_score(y_test, y_model_forest)))

## Métricas clásicas de Clasificadores
from sklearn import metrics
print(metrics.classification_report(y_model_forest, y_test))

## Matriz de confusión
from sklearn.metrics import confusion_matrix

mat = confusion_matrix(y_test, y_model_forest)

plt.close()

sns.heatmap(mat, square = True, annot = True, cbar = False)
plt.xlabel('Valor predicho')
plt.ylabel('Valor real')
```

Accuracy: 0.8905228758169934

	precision	recall	f1-score	support
0	0.98	0.93	0.96	430
1	0.75	0.85	0.79	143
2	0.59	0.57	0.58	35
3	0.50	0.67	0.57	3
4	0.00	0.00	0.00	1

accuracy			0.89	612
macro avg	0.56	0.60	0.58	612
weighted avg	0.90	0.89	0.89	612

[36]: Text(91.68, 0.5, 'Valor real')

