ML - XGBoost.

Levanta data_df.csv generado por EDA.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
# Lectura del dataset generado en el proceso de EDA.
df = pd.read_csv('data_df.csv')
# df=df.drop(['Unnamed: 0'], axis=1) # Elimino la columna que me generó el pd.write_csv() del E
```

In [3]:

```
from sklearn.model_selection import train_test_split
# !pip install xgboost
import xgboost as xgb
from xgboost import XGBClassifier ## Categorical target variable
# from xgboost import XGBRegressor ## Continuous target variable
```

In [4]:

```
# df.columns
```

In [5]:

```
# feature selection
# df con las variables independientes a incluir en el modelo. Surgen del EDA realizado.
# dfModel =df.drop('fraude', axis=1)
dfModel=df[['ase_antig_an', 'ase_cp', 'ase_codnac', 'pro_antig_an', 'modelo', 'anio', 'uso', 'c
```

In [6]:

```
# Es la variable dependiente/objetivo/target 'fraude'
dfTarget=df['fraude']
```

In [7]:

```
print("Cantidad Total: %.0f" % (dfTarget.count()))
print("Cantidad Fraudes: %.0f" % (dfTarget.sum()))
print("Fraudes: %.5f%%" % (dfTarget.sum() / dfTarget.count()))
```

```
Cantidad Total: 469370
Cantidad Fraudes: 17236
Fraudes: 0.03672%
```

In [8]:

```
# Divido en training y testing.
X_train, X_test, y_train, y_test = train_test_split(dfModel, dfTarget, test_size = 0.3, random_
```

In [9]:

```
print("Cantidad Training: %.0f" % (y_train.count()))
print("Cantidad Fraudes: %.0f" % (y_train.sum()))
print("Fraudes: %.5f%%" % (y_train.sum() / y_train.count()))
```

```
Cantidad Training: 328559
Cantidad Fraudes: 12140
Fraudes: 0.03695%
```

In [10]:

```
# Divido en training y testing.
# Utilizo stratify para mantener la proporción de positivos y negativos en train y test.
X_train, X_test, y_train, y_test = train_test_split(dfModel, dfTarget, test_size = 0.3, random_
```

In [11]:
```
print("Cantidad Training: %.0f" % (y_train.count()))
print("Cantidad Fraudes: %.0f" % (y_train.sum()))
print("Fraudes: %.5f%%" % (y_train.sum() / y_train.count()))
```

```
Cantidad Training: 328559
Cantidad Fraudes: 12065
Fraudes: 0.03672%
```

XGBOOST con parámetros estándard.

In [12]:
```
xgb_clas = XGBClassifier(use_label_encoder=False, colsample_bytree = 0.5, learning_rate = 0.2,
                         alpha = 10, n_estimators = 40)
```

In [13]:
```
# Fitting model
xgb_clas.fit(X_train, y_train)

# la variable que mas toma en cuenta el clasificador.
xgb.plot_importance(xgb_clas, ax=plt.gca())
predicted_data = xgb_clas.predict(X_test)

from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, predicted_data)

from sklearn.metrics import classification_report
print(classification_report(y_test, predicted_data))

xgb_clas.score(X_train, y_train) ## accuracy

from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score

accuracy = accuracy_score(y_test, predicted_data)
precision=precision_score(y_test, predicted_data)
recall=recall_score(y_test, predicted_data)
roc=roc_auc_score(y_test, predicted_data)
print("Accuracy: %.2f " % (accuracy))
print("Precision: %.2f " % (precision))
print("Recall: %.2f " % (recall))
print("AUC: %.2f " % (roc))

roc_curve(y_test, predicted_data)
```

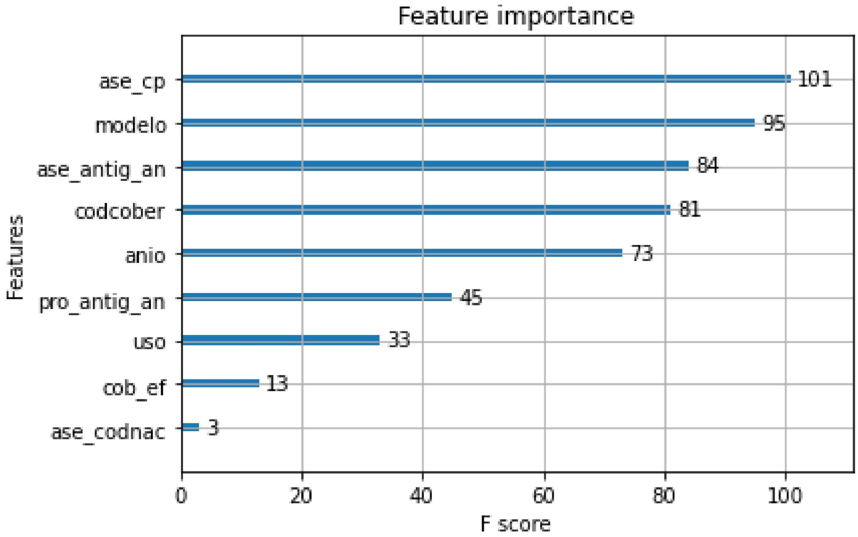
```
[09:31:59] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.c
c:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binar
y:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
              precision    recall  f1-score   support

           0       0.99      0.98      0.98    135640
           1       0.56      0.63      0.59      5171

    accuracy                           0.97    140811
   macro avg       0.77      0.80      0.79    140811
weighted avg       0.97      0.97      0.97    140811

Accuracy: 0.97
Precision: 0.56
Recall: 0.63
AUC: 0.80
```

Out[13]:
```
(array([0.        , 0.01921262, 1.        ]),
 array([0.        , 0.62869851, 1.        ]),
 array([2, 1, 0]))
```



Se agrega como métrica de evaluación 'auc'.

In [14]:

```
# agrego como métrica de evaluación 'auc'
xgb_clas = XGBClassifier(use_label_encoder=False, colsample_bytree = 0.5, learning_rate = 0.2,
                         alpha = 10, n_estimators = 40, eval_metric='auc')
```

In [15]:

```
# Fitting model
xgb_clas.fit(X_train, y_train)

# la variable que mas toma en cuenta el clasificador.
xgb.plot_importance(xgb_clas, ax=plt.gca())
predicted_data = xgb_clas.predict(X_test)

from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, predicted_data)

from sklearn.metrics import classification_report
print(classification_report(y_test, predicted_data))

xgb_clas.score(X_train, y_train) ## accuracy

from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score

accuracy = accuracy_score(y_test, predicted_data)
precision=precision_score(y_test, predicted_data)
recall=recall_score(y_test, predicted_data)
roc=roc_auc_score(y_test, predicted_data)
print("Accuracy: %.2f " % (accuracy))
print("Precision: %.2f " % (precision))
print("Recall: %.2f " % (recall))
print("AUC: %.2f " % (roc))

roc_curve(y_test, predicted_data)
```

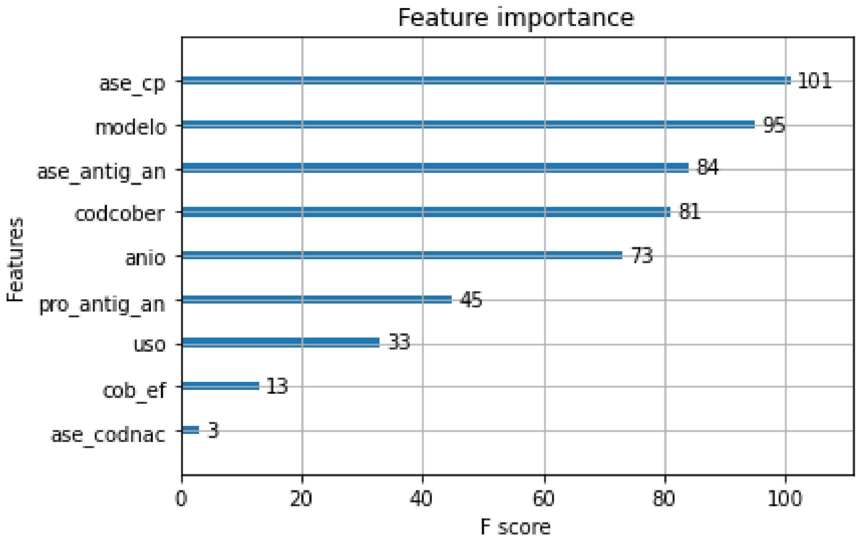
```
              precision    recall  f1-score   support

           0       0.99      0.98      0.98    135640
           1       0.56      0.63      0.59      5171

    accuracy                           0.97    140811
   macro avg       0.77      0.80      0.79    140811
weighted avg       0.97      0.97      0.97    140811

Accuracy: 0.97
Precision: 0.56
Recall: 0.63
AUC: 0.80
```

```
Out[15]: (array([0.        , 0.01921262, 1.        ]),
          array([0.        , 0.62869851, 1.        ]),
          array([2, 1, 0]))
```



BO.

```
In [16]:
def xgb_classifier(n_estimators, max_depth, reg_alpha, reg_lambda, min_child_weight, num_boost_
    params = {"booster": 'gbtree',
              "objective" : "binary:logistic",
              "eval_metric" : "auc",
              "is_unbalance": True,
              "n_estimators": int(n_estimators),
              "max_depth" : int(max_depth),
              "reg_alpha" : reg_alpha,
              "reg_lambda" : reg_lambda,
              "gamma": gamma,
              "num_threads" : 20,
              "min_child_weight" : int(min_child_weight),
              "learning_rate" : 0.02,
              "subsample_freq" : 5,
              "seed" : 42,
              "verbosity" : 0,
              "num_boost_round": int(num_boost_round)}

    train_data = xgb.DMatrix(X_train, y_train)

    cv_result = xgb.cv(params, train_data, 1000, early_stopping_rounds=100, stratified=True, nf
    return cv_result['test-auc-mean'].iloc[-1]
```

```
In [17]:
# !pip install bayesian-optimization
from bayes_opt import BayesianOptimization
xgbBO = BayesianOptimization(xgb_classifier, {  'n_estimators': (10, 100),
                                                'max_depth': (5, 40),
                                                'reg_alpha': (0.0, 0.1),
                                                'reg_lambda': (0.0, 0.1),
                                                'min_child_weight': (1, 10),
                                                'num_boost_round': (100, 1000),
                                                "gamma": (0, 10)
                                                })
```

```
In [18]:
xgbBO.maximize(n_iter=15, init_points=2)
```

```
|   iter    |  target   |   gamma   | max_depth | min_ch... | n_esti... | num_bo... | reg_alpha
| reg_la... |
-------------------------------------------------------------------------------------------------
--------------
|  1        |  0.9368   |  6.899    |  25.01    |  1.29     |  12.55    |  559.7    |  0.02048
|  0.06182  |
```

```
|  2        |  0.9356   |  2.19     |  17.93    |  1.07     |  67.56    |  718.7    |  0.03352  |
|  0.02529  |
|  3        |  0.9342   |  1.065    |  20.3     |  3.9      |  12.52    |  562.1    |  0.08613  |
|  0.0181   |
|  4        |  0.9365   |  3.751    |  15.6     |  9.446    |  42.11    |  805.1    |  0.02807  |
|  0.008394 |
|  5        |  0.9374   |  5.443    |  9.739    |  7.965    |  73.08    |  159.8    |  0.006044 |
|  0.08506  |
|  6        |  0.9373   |  8.318    |  16.65    |  7.291    |  81.9     |  565.9    |  0.09108  |
|  0.06751  |
|  7        |  0.9369   |  5.213    |  25.3     |  7.288    |  53.09    |  667.2    |  0.04166  |
|  0.05873  |
|  8        |  0.937    |  6.387    |  31.15    |  9.647    |  64.37    |  274.7    |  0.008601 |
|  0.0461   |
|  9        |  0.9348   |  1.576    |  26.66    |  4.229    |  69.99    |  341.1    |  0.04167  |
|  0.07295  |
|  10       |  0.9336   |  0.166    |  27.01    |  7.97     |  74.65    |  269.4    |  0.06958  |
|  0.0981   |
|  11       |  0.9375   |  4.137    |  7.08     |  5.901    |  54.37    |  852.2    |  0.04053  |
|  0.002811 |
|  12       |  0.9374   |  4.748    |  7.493    |  6.18     |  53.23    |  849.8    |  0.09236  |
|  0.09548  |
|  13       |  0.9371   |  2.573    |  10.03    |  4.735    |  49.41    |  853.3    |  0.006123 |
|  0.01085  |
|  14       |  0.9365   |  3.214    |  13.09    |  9.153    |  56.54    |  849.7    |  0.01228  |
|  0.01297  |
|  15       |  0.9374   |  6.299    |  11.1     |  7.818    |  78.72    |  165.3    |  0.04589  |
|  0.08864  |
|  16       |  0.9367   |  3.234    |  14.87    |  7.216    |  82.09    |  158.2    |  0.01261  |
|  0.001506 |
|  17       |  0.9372   |  9.12     |  16.76    |  8.848    |  71.73    |  164.5    |  0.06497  |
|  0.007996 |
===============================================================================================
==============
```

In [20]:

```python
xgbBO.max
```

Out[20]:

```
{'target': 0.937538,
 'params': {'gamma': 4.1365259441075,
  'max_depth': 7.080066619855001,
  'min_child_weight': 5.901283785299479,
  'n_estimators': 54.36852422030575,
  'num_boost_round': 852.2447035451108,
  'reg_alpha': 0.040529498872029494,
  'reg_lambda': 0.0028109131716275385}}
```

In [21]:

```python
xgb_clas = XGBClassifier(use_label_encoder=False,
                         colsample_bytree = 0.5,
                         learning_rate = 0.2,
                         gamma =  4.1365259441075,
                         max_depth = 7,
                         min_child_weight = 5.901283785299479,
                         alpha = 10,
                         n_estimators = 54,
                         reg_alpha =  0.040529498872029494,
                         reg_lambda = 0.0028109131716275385,
                         eval_metric='auc')
```

In [22]:

```python
# Fitting model
xgb_clas.fit(X_train, y_train)

# la variable que mas toma en cuenta el clasificador.
xgb.plot_importance(xgb_clas, ax=plt.gca())
predicted_data = xgb_clas.predict(X_test)

# dfAP=pd.DataFrame({'Actual': y_test, 'Prediccion': predicted_data})
# dfAP

from sklearn.metrics import confusion_matrix
```

```python
confusion_matrix(y_test, predicted_data)

from sklearn.metrics import classification_report
print(classification_report(y_test, predicted_data))

xgb_clas.score(X_train, y_train) ## accuracy

from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score

accuracy = accuracy_score(y_test, predicted_data)
precision=precision_score(y_test, predicted_data)
recall=recall_score(y_test, predicted_data)
roc=roc_auc_score(y_test, predicted_data)
print("Accuracy: %.2f " % (accuracy))
print("Precision: %.2f " % (precision))
print("Recall: %.2f " % (recall))
print("AUC: %.2f " % (roc))

roc_curve(y_test, predicted_data)
```

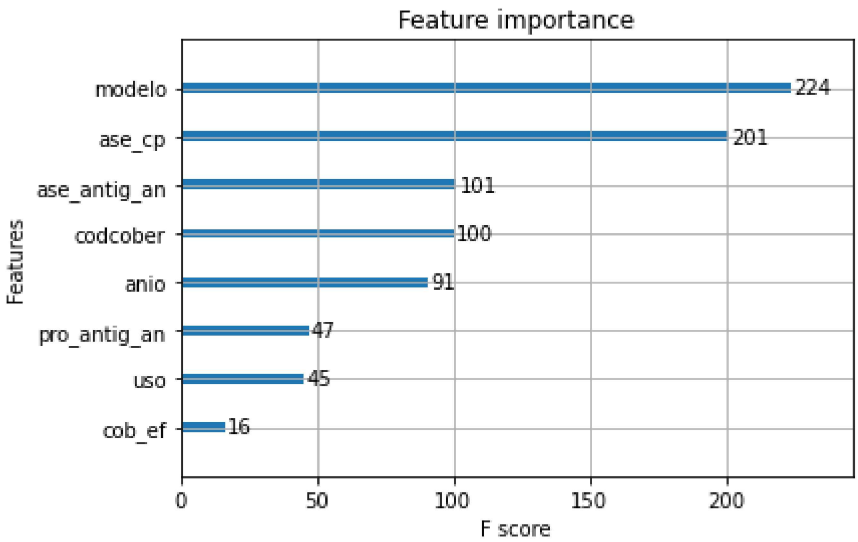
```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98    135640
           1       0.56      0.61      0.58      5171

    accuracy                           0.97    140811
   macro avg       0.77      0.79      0.78    140811
weighted avg       0.97      0.97      0.97    140811

Accuracy: 0.97
Precision: 0.56
Recall: 0.61
AUC: 0.79
```

Out[22]:
```
(array([0.        , 0.01848275, 1.        ]),
 array([0.        , 0.60549217, 1.        ]),
 array([2, 1, 0]))
```



In [23]:
```python
confusion_matrix(y_test, predicted_data)
```

Out[23]:
```
array([[133133,   2507],
       [  2040,   3131]], dtype=int64)
```

In [ ]: