Fraud Detection in Electricity and Gas Consumption Using XGBoost

April 2, 2024

1 Table of Contents

2 Authors

Mila Miletic, Jenny Louse, and Sergio Sanz.

3 GitHub Repository

https://github.com/sergio-sanz-rodriguez/Fraud-Detection-ML

4 Introduction

The Tunisian Company of Electricity and Gas (STEG) is a public and a non-administrative company, it is responsible for delivering electricity and gas across Tunisia. The company suffered tremendous losses in the order of 200 million Tunisian Dinars due to fraudulent manipulations of meters by consumers.

Using the client's billing history, the aim of the challenge is to detect and recognize clients involved in fraudulent activities. The solution will enhance the company's revenues and reduce the losses caused by such fraudulent activities.

In this notebook the potential of the **XBoost** classifier is evaluated for the detection of fraudulent cases.

For more information about this challenge, click here: https://zindi.africa/competitions/fraud-detection-in-electricity-and-gas-consumption-challenge

5 Download and Extract Files

[1]:
```
DATA_DIR = '/data'
TRAIN_DIR = f'{DATA_DIR}/train'
#TEST_DIR = f'{DATA_DIR}/test'
#OUTPUT_DIR = f'{DATA_DIR}/output'
```

[2]:
```
#import os.path
#from os import path
#
#for pth in [TRAIN_DIR, TEST_DIR, OUTPUT_DIR]:
#  if path.exists(pth) == False:
#    os.mkdir(pth)
```

[3]:
```
#only run this cell once, at the start
#import requests, os
#
#train_zip = "train.zip"
#test_zip = "test.zip"
#sample_sub = "SampleSubmission.csv"
```

[4]:
```
#!unzip "/content/train/train.zip" -d "/content/train/"
#!unzip "/content/test/test.zip" -d "/content/test/"
```

6 Import Libraries

[5]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import itertools
#import lightgbm
#from lightgbm import LGBMClassifier

import warnings
warnings.simplefilter('ignore')

# Modelling
from sklearn.preprocessing import StandardScaler, OneHotEncoder, MinMaxScaler,␣
 ↪RobustScaler
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
from sklearn.metrics import accuracy_score, recall_score, precision_score,␣
↪f1_score, roc_auc_score, fbeta_score, make_scorer, confusion_matrix,␣
↪classification_report, roc_curve
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV,␣
↪cross_val_predict
from sklearn.neighbors import KNeighborsClassifier
from scipy.stats import loguniform
from sklearn.pipeline import Pipeline
from imblearn.pipeline import Pipeline as ImbPipeline
from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer, KNNImputer
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from lightgbm import LGBMClassifier

import math

# Plot
import matplotlib.pyplot as plt
import seaborn as sns
#import plotly.express as px
from matplotlib.ticker import PercentFormatter
plt.rcParams.update({ "figure.figsize" : (8, 5),"axes.facecolor" : "white",␣
↪"axes.edgecolor":  "black"})
plt.rcParams["figure.facecolor"]= "w"
pd.plotting.register_matplotlib_converters()
pd.set_option('display.float_format', lambda x: '%.3f' % x)
pd.options.display.float_format = "{:,.2f}".format
import warnings
warnings.filterwarnings('ignore')

RSEED = 42
```

7 Evaluation Functions

[67]:
```
def predict_and_print_scores(model,
                             X_train,
                             y_train,
                             X_test,
```

```
                              y_test,
                              training=True,
                              test=True,
                              accuracy=True,
                              recall=True,
                              precision=True,
                              fbeta=[True, 1.0],
                              roc_auc=True,
                              matrix=True,
                              figsize=(3,2),
                              cmap='YlGn'):

    '''
    Given an already trained model, this function predicts and print some␣
↪performance scores training and/or testing data.
    The supported metrics are: accuracy, recall, precision, fbeta_score (and␣
↪f1_score if beta = 1.0), roc_auc.
    If the input parameter "matrix" is set to True, the function plot the␣
↪confusion matrix with a color map given in "cmap".

    model                Trained model
    X_train              Training data with features
    y_train              Training data with labels or targets
    X_test               Testing data with features
    y_test               Testing data with labels or targets                     ␣
↪
    training=True        True: print scores on the training set
    test=True            True: print scores on the testing set
    accuracy=True        True: print accuracy_score()
    recall=True          True: print recall_score()
    precision=True       True: print precision_score()
    fbeta=[True, 1.0]    [True, beta]: print fbeta_score. If beta = 1.0: f1_score
    roc_auc=True         True: print roc_auc_score()
    matrix=True          True: plot confusion matrix
    figsize=(3,2)        Figure size for the confusion matrix
    cmap='YlGn')         Color map for the confusion matrix

    Possible color maps: 'Greys', 'Purples', 'Blues', 'Greens', 'Oranges',␣
↪'Reds',
                           'YlOrBr', 'YlOrRd', 'OrRd', 'PuRd', 'RdPu', 'BuPu',
                           'GnBu', 'PuBu', 'YlGnBu', 'PuBuGn', 'BuGn', 'YlGn'

    Returns: fig, ax: the figure objects of the confusion matrix (if enabled)
    '''

    # Prediction
    y_pred_train = model.predict(X_train)
```

```
    y_pred_test = model.predict(X_test)

    # Scores
    if accuracy:
        if training:
            print("Accuracy on training set:", round(accuracy_score(y_train,␣
↪y_pred_train), 2))
        if test:
            print("Accuracy on test set:", round(accuracy_score(y_test,␣
↪y_pred_test), 2))
        print("--------"*5)

    if recall:
        if training:
            print("Recall on training set:", round(recall_score(y_train,␣
↪y_pred_train), 2))
        if test:
            print("Recall on test set:", round(recall_score(y_test,␣
↪y_pred_test), 2))
        print("--------"*5)

    if precision:
        if training:
            print("Precision on training set:", round(precision_score(y_train,␣
↪y_pred_train), 2))
        if test:
            print("Precision on test set:", round(precision_score(y_test,␣
↪y_pred_test), 2))
        print("--------"*5)

    if fbeta[0]:
        if training:
            print("fbeta_score on training set:", round(fbeta_score(y_train,␣
↪y_pred_train, beta=fbeta[1]), 2))
        if test:
            print("fbeta_score on test set:", round(fbeta_score(y_test,␣
↪y_pred_test, beta=fbeta[1]), 2))
        print("--------"*5)

    if roc_auc:
        y_pred_train_p = model.predict_proba(X_train)[:,1]
        y_pred_test_p = model.predict_proba(X_test)[:,1]
        if training:
            print('roc_auc_score on training set: ',␣
↪round(roc_auc_score(y_train, y_pred_train_p), 2))
        if test:
```

```
            print('roc_auc_score on test set: ', round(roc_auc_score(y_test,␣
 ↪y_pred_test_p), 2))
        print("--------"*5)

    # Plot confusion matrix
    if matrix:
        fig = plt.figure(figsize=figsize)
        ax = fig.add_subplot()
        sns.heatmap(confusion_matrix(y_test, y_pred_test), annot=True,␣
 ↪cmap=cmap);
        plt.title('Test Set')
        plt.ylabel('True label')
        plt.xlabel('Predicted label')

        return fig, ax

def find_roc_threshold_tpr(model, X, y, value_target):

    """
    This function calculates the threshold and false positive rate␣
 ↪corresponding to a true positive rate of value_target (from 0 to 1).

    model                  # Trained model
    X                      # Feature dataset
    y                      # Target dataset
    value_target           # True positive rate value

    Returns:

    threshold              # Threshold value
    false_positive_rate    # False positive rate value
    """

    fpr, tpr, thr = roc_curve(y, model.predict_proba(X)[:,1])

    old_diff = 100000000
    for index, value in enumerate(tpr):
        new_diff = abs(value_target - value)
        if new_diff < old_diff:
            false_pos_rate = fpr[index]
            threshold = thr[index]
            old_diff = new_diff

    return threshold, false_pos_rate

def find_roc_threshold_f1(model, X, y):
```

```
    """
    This function calculates the threshold in the ROC curve that maximizes the␣
 ↪f1 score.
    model                   # Trained model
    X                       # Feature dataset
    y                       # Target dataset

    Returns:

    best_threshold          # Threshold value
    best_f1_score           # False positive rate value

    """

    pred_ = model.predict_proba(X)[:,1]

    best_threshold = 0.5
    best_f1_score = 0.0
    for value in np.arange(1, 10, 0.5):
        pred_tmp = np.where(pred_ >= float(value/10), 1, 0)
        cost = f1_score(y, pred_tmp)
        if cost > best_f1_score:
            best_f1_score = cost
            best_threshold = float(value/10)

    return best_threshold, best_f1_score

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Oranges,
                          figsize=(10,10)):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    Source: http://scikit-learn.org/stable/auto_examples/model_selection/
 ↪plot_confusion_matrix.html
    """
    # Confusion matrix
    #cm = confusion_matrix(test_labels, rf_predictions)
    #plot_confusion_matrix(cm, classes = ['Poor Health', 'Good Health'],
    #                      title = 'Health Confusion Matrix')

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
```

```python
        print('Confusion matrix, without normalization')

    # Plot the confusion matrix
    plt.figure(figsize = figsize)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title, size = 18)
    plt.colorbar(aspect=4)
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45, size = 14)
    plt.yticks(tick_marks, classes, size = 14)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.

    # Labeling the plot
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt), fontsize = 18,
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label', size = 18)
    plt.xlabel('Predicted label', size = 18)

def plot_roc_curves(model_dic, X_test, y_test, figsize=(6,5)):

    """
    This function plots the ROC curves of the models defined in model_dic.
    The model_dic format is {'model_label' : [model_object, color-line'], ...}.␣
 ↪Example:
    model_dic = {['model_1' : [model_1, 'r-'], 'model_2' : [model_2, 'b-']}
    """

    fig = plt.figure(figsize=figsize)
    ax = fig.add_subplot()

    for key, _ in model_dic.items():

        model = model_dic[key][0]

        fpr, tpr, _ = roc_curve(y_test, model.predict_proba(X_test)[:,1])
        plt.plot(fpr, tpr, model_dic[key][1], label=key)

    plt.plot([0,1],[0,1],'k:',label='Random')
    plt.plot([0,0,1,1],[0,1,1,1],'k--',label='Perfect')
    ax.set_xlabel('False Positive Rate (1 - Specifity)')
    ax.set_ylabel('True Positive Rate (Recall)')
```

```
    plt.title('ROC Curve', size = 18)
    ax.legend()
    plt.grid()
    plt.show()

    return fig, ax
```

8 Data Preparation

8.1 Read the Data

[7]:
```
client = pd.read_csv('data/train/client_train.csv', low_memory=False)
invoice = pd.read_csv('data/train/invoice_train.csv', low_memory=False)
#client_test = pd.read_csv('data/test/client_test.csv', low_memory=False)
#invoice_test = pd.read_csv('data/test/invoice_test.csv', low_memory=False)
```

Columns:

- Client_id: Unique id for client
- District: District where the client is
- Client_catg: Category client belongs to
- Region: Area where the client is
- Creation_date: Date client joined
- Target: fraud:1 , not fraud: 0
- Tarif_type: Type of tax
- Counter_statue: takes up to 5 values such as working fine, not working, on hold statue, ect
- Counter_code:
- Reading_remarque: notes that the STEG agent takes during his visit to the client (e.g: If the counter shows something wrong, the agent gives a bad score)
- Counter_coefficient: An additional coefficient to be added when standard consumption is exceeded
- Consommation_level_1: Consumption_level_1
- Consommation_level_2: Consumption_level_2
- Consommation_level_3: Consumption_level_3
- Consommation_level_4: Consumption_level_4
- Months_number: Month number
- Counter_type: Type of counter

8.2 Data Understanding

[8]:
```
#compare size of the various datasets
print(client.shape, invoice.shape)
```

```
(135493, 6) (4476749, 16)
```

[9]:
```
#print top rows of dataset
invoice.head(2)
```

[9]:
```
        client_id invoice_date  tarif_type  counter_number counter_statue
0  train_Client_0   2014-03-24          11         1335667              0  \
1  train_Client_0   2013-03-29          11         1335667              0

   counter_code  reading_remarque  counter_coefficient  consommation_level_1
0           203                 8                    1                    82  \
1           203                 6                    1                  1200

   consommation_level_2  consommation_level_3  consommation_level_4
0                     0                     0                     0  \
1                   184                     0                     0

   old_index  new_index  months_number counter_type
0      14302      14384              4         ELEC
1      12294      13678              4         ELEC
```

[10]:
```
client.head(2)
```

[10]:
```
   disrict       client_id  client_catg  region creation_date  target
0       60  train_Client_0           11     101    31/12/1994    0.00
1       69  train_Client_1           11     107    29/05/2002    0.00
```

[11]:
```
#Get a summary for all numerical columns
invoice.describe().T
```

[11]:
```
                            count                 mean                   std
tarif_type           4,476,749.00                20.13                 13.47  \
counter_number       4,476,749.00 123,058,699,065.18 1,657,267,274,261.93
counter_code         4,476,749.00               172.49                133.89
reading_remarque     4,476,749.00                 7.32                  1.57
counter_coefficient  4,476,749.00                 1.00                  0.31
consommation_level_1 4,476,749.00               410.98                757.31
consommation_level_2 4,476,749.00               109.32              1,220.12
consommation_level_3 4,476,749.00                20.31                157.42
consommation_level_4 4,476,749.00                52.93                875.47
old_index            4,476,749.00            17,767.00             40,366.93
new_index            4,476,749.00            18,349.70             40,953.21
months_number        4,476,749.00                44.83              3,128.34
```

```
                        min        25%        50%          75%  
tarif_type             8.00      11.00      11.00        40.00  \
counter_number         0.00 121,108.00 494,561.00 1,115,161.00   
counter_code           0.00       5.00     203.00       207.00   
reading_remarque       5.00       6.00       8.00         9.00   
counter_coefficient    0.00       1.00       1.00         1.00   
consommation_level_1   0.00      79.00     274.00       600.00   
consommation_level_2   0.00       0.00       0.00         0.00   
consommation_level_3   0.00       0.00       0.00         0.00   
consommation_level_4   0.00       0.00       0.00         0.00   
old_index              0.00   1,791.00   7,690.00    21,660.00   
new_index              0.00   2,056.00   8,192.00    22,343.00   
months_number          0.00       4.00       4.00         4.00   

                                      max  
tarif_type                          45.00  
counter_number       27,981,145,458,733.00  
counter_code                       600.00  
reading_remarque                   413.00  
counter_coefficient                 50.00  
consommation_level_1           999,910.00  
consommation_level_2           999,073.00  
consommation_level_3            64,492.00  
consommation_level_4           547,946.00  
old_index                    2,800,280.00  
new_index                    2,870,972.00  
months_number                  636,624.00  
```

[12]:
```python
#Get a summary for all numerical columns
client.describe()
```

[12]:

	disrict	client_catg	region	target
count	135,493.00	135,493.00	135,493.00	135,493.00
mean	63.51	11.51	206.16	0.06
std	3.35	4.42	104.21	0.23
min	60.00	11.00	101.00	0.00
25%	62.00	11.00	103.00	0.00
50%	62.00	11.00	107.00	0.00
75%	69.00	11.00	307.00	0.00
max	69.00	51.00	399.00	1.00

[13]:
```python
#Get concise information of each column in dataset
invoice.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4476749 entries, 0 to 4476748
Data columns (total 16 columns):
```

```
 #   Column                 Dtype
---  ------                 -----
 0   client_id              object
 1   invoice_date           object
 2   tarif_type             int64
 3   counter_number         int64
 4   counter_statue         object
 5   counter_code           int64
 6   reading_remarque       int64
 7   counter_coefficient    int64
 8   consommation_level_1   int64
 9   consommation_level_2   int64
 10  consommation_level_3   int64
 11  consommation_level_4   int64
 12  old_index              int64
 13  new_index              int64
 14  months_number          int64
 15  counter_type           object
dtypes: int64(12), object(4)
memory usage: 546.5+ MB
```

[14]:
```
#Get concise information of each column in dataset
client.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 135493 entries, 0 to 135492
Data columns (total 6 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   disrict         135493 non-null  int64
 1   client_id       135493 non-null  object
 2   client_catg     135493 non-null  int64
 3   region          135493 non-null  int64
 4   creation_date   135493 non-null  object
 5   target          135493 non-null  float64
dtypes: float64(1), int64(3), object(2)
memory usage: 6.2+ MB
```

[15]:
```
#Getting unique values on the invoice train data
for col in invoice.columns:
    print(f"{col} - {invoice[col].nunique()}")
```

```
client_id - 135493
invoice_date - 8275
tarif_type - 17
counter_number - 201893
counter_statue - 12
counter_code - 42
reading_remarque - 8
```

```
counter_coefficient - 16
consommation_level_1 - 8295
consommation_level_2 - 12576
consommation_level_3 - 2253
consommation_level_4 - 12075
old_index - 155648
new_index - 157980
months_number - 1370
counter_type - 2
```

[16]:
```python
#Getting unique values on the invoice train data
for col in client.columns:
    print(f"{col} - {client[col].nunique()}")
```

```
disrict - 4
client_id - 135493
client_catg - 3
region - 25
creation_date - 8088
target - 2
```

[17]:
```python
#check for missing values
invoice.isnull().sum()
```

[17]:
```
client_id               0
invoice_date            0
tarif_type              0
counter_number          0
counter_statue          0
counter_code            0
reading_remarque        0
counter_coefficient     0
consommation_level_1    0
consommation_level_2    0
consommation_level_3    0
consommation_level_4    0
old_index               0
new_index               0
months_number           0
counter_type            0
dtype: int64
```

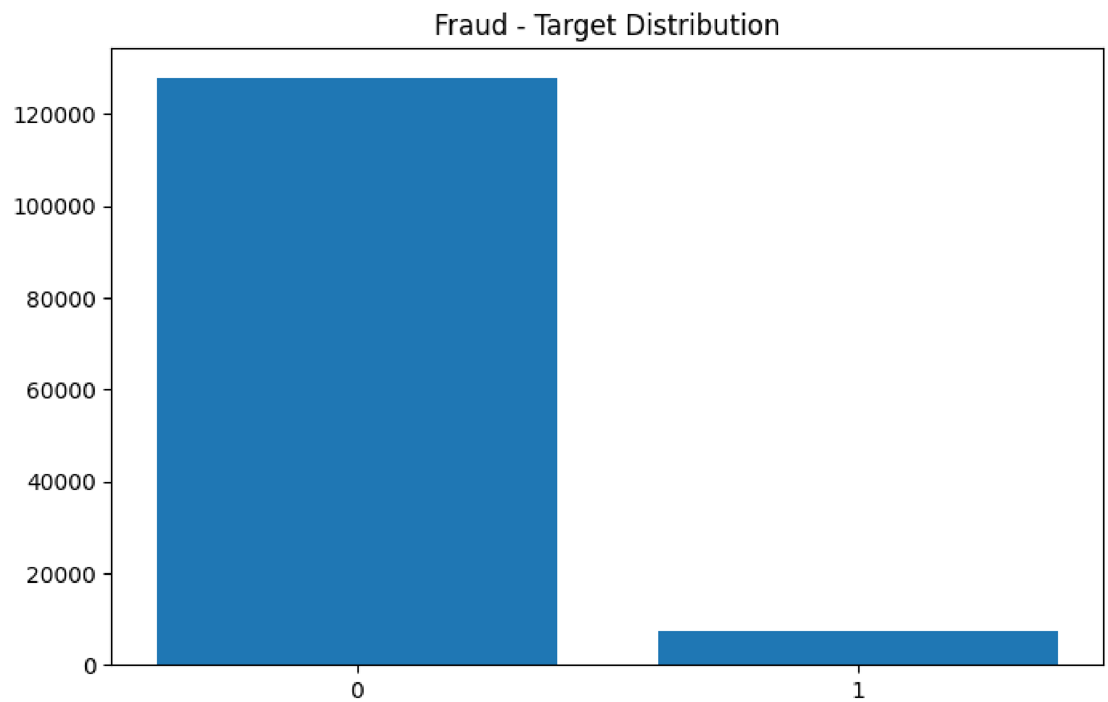
[18]:
```python
#check for missing values
client.isnull().sum()
```

[18]:
```
disrict          0
client_id        0
client_catg      0
```

```
region           0
creation_date    0
target           0
dtype: int64
```

No missing values in train set

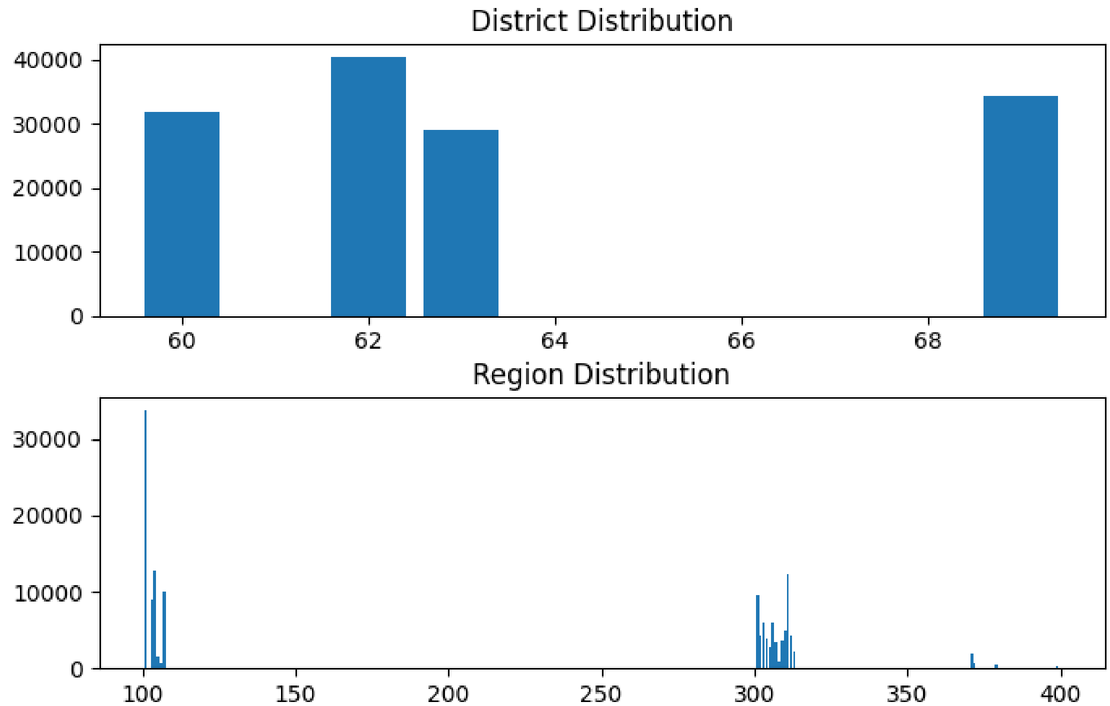
[19]:
```
#Visualize fraudulent activities
fraudactivities = client.groupby(['target'])['client_id'].count()
plt.bar(x=fraudactivities.index, height=fraudactivities.values, tick_label =
 ↪[0,1])
plt.title('Fraud - Target Distribution')
plt.show()
```



The target is highly imbalanced with fewer cases of fraudulent activities. Oversampling or undersampling methods will be investigated.

[20]:
```
#Visualize client distribution across districts and regions
region1 = client.groupby('disrict')['client_id'].count()
fig = plt.figure()
ax1 = fig.add_subplot(211)
plt.bar(x=region1.index, height=region1.values)
plt.title('District Distribution')
region2 = client.groupby('region')['client_id'].count()
ax2 = fig.add_subplot(212)
```

```
plt.bar(x=region2.index, height=region2.values)
plt.title('Region Distribution')
plt.subplots_adjust(hspace=0.3)
plt.show()
```



9 Feature Engineering

[21]:
```
#rename columns
client.rename(columns={
    'disrict': 'district',
}, inplace=True)

invoice.rename(columns={
    'counter_statue': 'counter_status',
    'reading_remarque': 'agent_remark',
    # Add more columns as needed
}, inplace=True)
```

[22]:
```
# List of columns to convert to categorical
columns_to_convert = ['tarif_type', 'counter_code', 'months_number',␣
↪'counter_type']

# Convert each column in invoice
```

```
for column in columns_to_convert:
    invoice[column] = invoice[column].astype('category')
```

[23]:
```
# List of columns to convert to categorical
columns_to_convert = ['client_id', 'region','district']

# Convert each column in client
for column in columns_to_convert:
    client[column] = client[column].astype('category')
```

[24]:
```
# Convert columns to integer, ensuring support for NaN values
columns_to_convert = ['target']

# Convert each column in client to a pandas nullable integer type
for column in columns_to_convert:
    client[column] = client[column].astype('int32')
```

[25]:
```
# Change strings in counter_status to integers
def convert_to_int(value):
    # Check if the value is 'A' and return 500
    if value == 'A':
        return 500
    # Try to convert numeric strings directly to int
    try:
        return int(value)
    # If conversion fails (which shouldn't happen with the given conditions),␣
↪return the value
    except ValueError:
        return value

invoice['counter_status'] = invoice['counter_status'].apply(convert_to_int)
```

[26]:
```
# convert columns to integer, ensuring support for NaN values
columns_to_convert = ['counter_status','counter_code']

# Convert each column in client to a pandas nullable integer type
for column in columns_to_convert:
    invoice[column] = invoice[column].astype('int32')
```

[27]:
```
#Change date to datetime
client['creation_date'] = pd.to_datetime(client['creation_date'])

#Change date to datetime
invoice['invoice_date'] = pd.to_datetime(invoice['invoice_date'])
```

[28]:
```
# calculate total consumption per billing cycle per counter type
```

```
invoice['total_consumption'] = invoice[['consommation_level_1',␣
↪'consommation_level_2', 'consommation_level_3', 'consommation_level_4']].
↪sum(axis=1)
```

[29]:
```
# Then, aggregate total_consumption by client_id
aggregated_consumption_ener = invoice.groupby('client_id')['total_consumption'].
↪agg(
    ener_total_consumption='sum',  # Aggregate the total
    ener_min_consumption='min',
    ener_max_consumption='max',
    ener_mean_consumption='mean',
    ener_std_consumption='std',
    ener_range_consumption=lambda x: x.max() - x.min()  # Calculate the range␣
↪as max - min
).reset_index()
```

[30]:
```
invoice['invoice_month'] = invoice['invoice_date'].dt.month
invoice['invoice_year'] = invoice['invoice_date'].dt.year
```

[31]:
```
# Aggregate consumption by client_id for each consumption level separately
aggregated_consumption_ener_1 = invoice.
↪groupby('client_id')['consommation_level_1'].agg(
    cons1_total='sum',
    cons1_min='min',
    cons1_max='max',
    cons1_mean='mean',
    cons1_std='std',
    cons1_range=lambda x: x.max() - x.min()
).reset_index()

aggregated_consumption_ener_2 = invoice.
↪groupby('client_id')['consommation_level_2'].agg(
    cons2_total='sum',
    cons2_min='min',
    cons2_max='max',
    cons2_mean='mean',
    cons2_std='std',
    cons2_range=lambda x: x.max() - x.min()
).reset_index()

aggregated_consumption_ener_3 = invoice.
↪groupby('client_id')['consommation_level_3'].agg(
    cons3_total='sum',
    cons3_min='min',
    cons3_max='max',
    cons3_mean='mean',
    cons3_std='std',
```

```
    cons3_range=lambda x: x.max() - x.min()
).reset_index()

aggregated_consumption_ener_4 = invoice.
 ↪groupby('client_id')['consommation_level_4'].agg(
    cons4_total='sum',
    cons4_min='min',
    cons4_max='max',
    cons4_mean='mean',
    cons4_std='std',
    cons4_range=lambda x: x.max() - x.min()
).reset_index()
```

[32]:
```
# Replace values of counter_status
invoice['counter_status'] = invoice['counter_status'].replace(500, 6)
invoice['counter_status'] = invoice['counter_status'].replace(769, 7)
invoice['counter_status'] = invoice['counter_status'].replace(618, 8)
invoice['counter_status'] = invoice['counter_status'].replace(269375, 9)
invoice['counter_status'] = invoice['counter_status'].replace(46, 10)
invoice['counter_status'] = invoice['counter_status'].replace(420, 11)
```

[33]:
```
# Aggregate counter_status by client_id
aggregated_counter_status = invoice.groupby('client_id')['counter_status'].agg(
    counter_status_min='min',
    counter_status_max='max',
    counter_status_mean='mean',
    counter_status_std='std',
).reset_index()
```

[34]:
```
# Replace values of agent_remark
invoice['agent_remark'] = invoice['agent_remark'].replace(5, 1)
invoice['agent_remark'] = invoice['agent_remark'].replace(6, 2)
invoice['agent_remark'] = invoice['agent_remark'].replace(7, 3)
invoice['agent_remark'] = invoice['agent_remark'].replace(8, 4)
invoice['agent_remark'] = invoice['agent_remark'].replace(9, 5)
invoice['agent_remark'] = invoice['agent_remark'].replace(203, 6)
invoice['agent_remark'] = invoice['agent_remark'].replace(207, 7)
invoice['agent_remark'] = invoice['agent_remark'].replace(413, 8)
```

[35]:
```
# Aggregate agent_remark by client_id
aggregated_agent_remark = invoice.groupby('client_id')['agent_remark'].agg(
    agent_remark_min='min',
    agent_remark_max='max',
    agent_remark_mean='mean',
    agent_remark_std='std',
    #agent_remark_mode='mode'
).reset_index()
```

[36]:
```python
# Aggregate counter_coefficient by client_id
aggregated_counter_coefficient = invoice.
↪groupby('client_id')['counter_coefficient'].agg(
    counter_coefficient_min='min',
    counter_coefficient_max='max',
    counter_coefficient_mean='mean',
    counter_coefficient_std='std',
   # counter_coefficient_remark_mode='mode'
).reset_index()
```

[37]:
```python
# Aggregate counter_code by client_id
aggregated_counter_code = invoice.groupby('client_id')['counter_code'].agg(
    counter_code_min='min',
    counter_code_max='max',
    counter_code_mean='mean',
    counter_code_std='std',
   # counter_coefficient_remark_mode='mode'
).reset_index()
```

[38]:
```python
# Create transaction_count feature
grouped_counts = invoice.groupby('client_id').size().
↪reset_index(name='transaction_count')
```

[39]:
```python
# Sort invoice DataFrame by 'client_id', 'counter_type', and 'invoice_date'
invoice_sorted = invoice.sort_values(['client_id', 'counter_type',␣
↪'invoice_date'])

# Calculate the difference in days between invoice dates within each group of␣
↪'client_id' and 'counter_type'
invoice_sorted['invoice_delta_time'] = invoice_sorted.groupby(['client_id',␣
↪'counter_type'])['invoice_date'].diff().dt.days

# Create a new DataFrame focusing on the columns of interest
date_eda = invoice_sorted[['client_id', 'counter_type', 'invoice_date',␣
↪'invoice_delta_time']].copy()

# Sort this new DataFrame by 'client_id', 'counter_type', and 'invoice_date'
date_eda_sorted = date_eda.sort_values(['client_id', 'counter_type',␣
↪'invoice_date'])
```

[40]:
```python
# Sort invoice DataFrame by 'client_id' and 'invoice_date'
invoice_sorted = invoice.sort_values(['client_id', 'invoice_date'])

# Calculate the difference in days between invoice dates within each group of␣
↪'client_id' and 'counter_type'
invoice_sorted['invoice_delta_time'] = invoice_sorted.
↪groupby(['client_id'])['invoice_date'].diff().dt.days
```

```
# Create a new DataFrame focusing on the columns of interest
date_eda = invoice_sorted[['client_id', 'invoice_date', 'invoice_delta_time']].
 ↪copy()

# Sort this new DataFrame by 'client_id', 'counter_type', and 'invoice_date'
date_eda_sorted = date_eda.sort_values(['client_id', 'invoice_date'])
```

[41]:
```
# Group by 'client_id' and then calculate the aggregate statistics for␣
 ↪'invoice_delta_time'
aggregated_ener_date_stats = date_eda_sorted.
 ↪groupby(['client_id'])['invoice_delta_time'].agg(
    ener_min_invoice_delta='min',
    ener_max_invoice_delta='max',
    ener_mean_invoice_delta='mean',
    #elec_median_invoice_delta='median',
    ener_std_invoice_delta='std'
).reset_index()
```

[42]:
```
# List of columns you want to include in the new DataFrame
columns_to_include = ['client_id', 'client_catg', 'region', 'creation_date',␣
 ↪'target']

# Create a new DataFrame with the specified columns
model_df = client[columns_to_include].copy()
```

[43]:
```
# Create the whole dataframe
model_df = model_df.merge(aggregated_counter_status[['client_id',
                                                     'counter_status_min',
                                                     'counter_status_max',
                                                     'counter_status_mean',
                                                     'counter_status_std']],␣
 ↪on='client_id', how='left')

model_df = model_df.merge(aggregated_agent_remark[['client_id',
                                                   'agent_remark_min',
                                                   'agent_remark_max',
                                                   'agent_remark_mean',
                                                   'agent_remark_std']],␣
 ↪on='client_id', how='left')

model_df = model_df.merge(aggregated_counter_coefficient[['client_id',
                                                                         ␣
 ↪'counter_coefficient_min',
                                                                         ␣
 ↪'counter_coefficient_max',
```

```
↪'counter_coefficient_mean',
                                                                        ␣
↪'counter_coefficient_std']], on='client_id', how='left')

model_df = model_df.merge(aggregated_counter_code[['client_id',
                                                        'counter_code_min',
                                                        'counter_code_max',
                                                        'counter_code_mean',
                                                        'counter_code_std']],␣
↪on='client_id', how='left')

model_df = model_df.merge(grouped_counts[['client_id','transaction_count']],␣
↪on='client_id', how='left')
```

[44]:
```
model_df = model_df.merge(aggregated_consumption_ener[['client_id',
                                                      'ener_total_consumption',
                                                      'ener_min_consumption',
                                                      'ener_max_consumption',
                                                      'ener_mean_consumption',
                                                                        ␣
↪'ener_std_consumption']], on='client_id', how='left')

model_df = model_df.merge(aggregated_ener_date_stats[['client_id',
                                                     'ener_min_invoice_delta',
                                                     'ener_max_invoice_delta',
                                                     'ener_mean_invoice_delta',
                                                                        ␣
↪'ener_std_invoice_delta']], on='client_id', how='left')

model_df = model_df.merge(aggregated_consumption_ener_1[['client_id',
                                                        'cons1_total',
                                                        'cons1_min',
                                                        'cons1_max',
                                                        'cons1_mean',
                                                        'cons1_std',
                                                        'cons1_range']],␣
↪on='client_id', how='left')


model_df = model_df.merge(aggregated_consumption_ener_2[['client_id',
                                                        'cons2_total',
                                                        'cons2_min',
                                                        'cons2_max',
                                                        'cons2_mean',
                                                        'cons2_std',
```

```
                                                            'cons2_range']],␣
↪on='client_id', how='left')


model_df = model_df.merge(aggregated_consumption_ener_3[['client_id',
                                                            'cons3_total',
                                                            'cons3_min',
                                                            'cons3_max',
                                                            'cons3_mean',
                                                            'cons3_std',
                                                            'cons3_range']],␣
↪on='client_id', how='left')


model_df = model_df.merge(aggregated_consumption_ener_4[['client_id',
                                                            'cons4_total',
                                                            'cons4_min',
                                                            'cons4_max',
                                                            'cons4_mean',
                                                            'cons4_std',
                                                            'cons4_range']],␣
↪on='client_id', how='left')
```

[45]:
```
# Compute contract duration
model_df['coop_time'] = (2019 - model_df['creation_date'].dt.year)*12 -␣
↪model_df['creation_date'].dt.month
```

[46]:
```
# Compute invoice per cooperation
model_df['invoice_per_cooperation'] = model_df['transaction_count'] /␣
↪model_df['coop_time']
model_df['invoice_per_cooperation'].replace([np.inf, -np.inf], 0, inplace=True)
```

[47]:
```
# Convert numerical features of type int64 to int32
for column in model_df.columns:
    if model_df[column].dtype == 'int64':
        model_df[column] = model_df[column].astype('int32')
```

10 Machine-Learning Modeling

[48]:
```
# Drop redundant columns
model_df_copy = model_df.copy()
drop_columns = ['client_id', 'creation_date']

for col in drop_columns:
    if col in model_df_copy.columns:
        model_df_copy.drop([col], axis=1, inplace=True)
```

[49]:
```python
# Print the list of final features and their types
model_df_copy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 135493 entries, 0 to 135492
Data columns (total 55 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   client_catg               135493 non-null  int32
 1   region                    135493 non-null  category
 2   target                    135493 non-null  int32
 3   counter_status_min        135493 non-null  int32
 4   counter_status_max        135493 non-null  int32
 5   counter_status_mean       135493 non-null  float64
 6   counter_status_std        131281 non-null  float64
 7   agent_remark_min          135493 non-null  int32
 8   agent_remark_max          135493 non-null  int32
 9   agent_remark_mean         135493 non-null  float64
 10  agent_remark_std          131281 non-null  float64
 11  counter_coefficient_min   135493 non-null  int32
 12  counter_coefficient_max   135493 non-null  int32
 13  counter_coefficient_mean  135493 non-null  float64
 14  counter_coefficient_std   131281 non-null  float64
 15  counter_code_min          135493 non-null  int32
 16  counter_code_max          135493 non-null  int32
 17  counter_code_mean         135493 non-null  float64
 18  counter_code_std          131281 non-null  float64
 19  transaction_count         135493 non-null  int32
 20  ener_total_consumption    135493 non-null  int32
 21  ener_min_consumption      135493 non-null  int32
 22  ener_max_consumption      135493 non-null  int32
 23  ener_mean_consumption     135493 non-null  float64
 24  ener_std_consumption      131281 non-null  float64
 25  ener_min_invoice_delta    131281 non-null  float64
 26  ener_max_invoice_delta    131281 non-null  float64
 27  ener_mean_invoice_delta   131281 non-null  float64
 28  ener_std_invoice_delta    125906 non-null  float64
 29  cons1_total               135493 non-null  int32
 30  cons1_min                 135493 non-null  int32
 31  cons1_max                 135493 non-null  int32
 32  cons1_mean                135493 non-null  float64
 33  cons1_std                 131281 non-null  float64
 34  cons1_range               135493 non-null  int32
 35  cons2_total               135493 non-null  int32
 36  cons2_min                 135493 non-null  int32
 37  cons2_max                 135493 non-null  int32
 38  cons2_mean                135493 non-null  float64
 39  cons2_std                 131281 non-null  float64
```

```
 40   cons2_range                 135493 non-null   int32
 41   cons3_total                 135493 non-null   int32
 42   cons3_min                   135493 non-null   int32
 43   cons3_max                   135493 non-null   int32
 44   cons3_mean                  135493 non-null   float64
 45   cons3_std                   131281 non-null   float64
 46   cons3_range                 135493 non-null   int32
 47   cons4_total                 135493 non-null   int32
 48   cons4_min                   135493 non-null   int32
 49   cons4_max                   135493 non-null   int32
 50   cons4_mean                  135493 non-null   float64
 51   cons4_std                   131281 non-null   float64
 52   cons4_range                 135493 non-null   int32
 53   coop_time                   135493 non-null   int32
 54   invoice_per_cooperation     135493 non-null   float64
dtypes: category(1), float64(23), int32(31)
memory usage: 39.9 MB
```

10.1 Train - Test Split

[50]:
```
y = model_df_copy['target']
X = model_df_copy.drop('target', axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=42, stratify=y)
X_train.shape, X_test.shape
```

[50]:
```
((108394, 54), (27099, 54))
```

10.2 Logistic Regressor (Baseline)

[51]:
```
# Separate features by type
num_features = X_train.select_dtypes(include=['int32', 'float64']).columns
cat_features = X_train.select_dtypes(include=['object', 'category']).columns

# Preprocessing for numerical data
num_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value=0)),
    ('scale', StandardScaler())
])

# Preprocessing for categorical data
cat_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value=0)),
    ('onehot', OneHotEncoder(drop='first'))
])
```

```
# Apply preprocessing to num and categorical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', num_transformer, num_features),
        ('cat', cat_transformer, cat_features)
    ],
    remainder='passthrough')

model_lg = ImbPipeline([
    ('prep', preprocessor),
    ('undersample', RandomUnderSampler(sampling_strategy=0.5)),
    ('lg', LogisticRegression())
])

model_lg.fit(X_train, y_train)

predict_and_print_scores(model_lg, X_train, y_train, X_test, y_test,␣
 ↪matrix=False)
```

```
Accuracy on training set: 0.86
Accuracy on test set: 0.86
----------------------------------------
Recall on training set: 0.36
Recall on test set: 0.37
----------------------------------------
Precision on training set: 0.16
Precision on test set: 0.17
----------------------------------------
fbeta_score on training set: 0.22
fbeta_score on test set: 0.23
----------------------------------------
roc_auc_score on training set:  0.75
roc_auc_score on test set:  0.75
----------------------------------------
```

10.3 XGBoost Classifier with Full Grid Search

[52]:
```
# Build the pipeline and train

#xgb_pipe = ImbPipeline([
#    ('prep', preprocessor),
#    ('undersample', RandomUnderSampler(sampling_strategy=0.5)),
#    ('xgb', XGBClassifier(use_label_encoder=False, eval_metric='logloss'))
#])

#hyperparameters = {
#    'xgb__learning_rate': [0.001, 0.01, 0.1, 0.5, 1],
#    'xgb__n_estimators': [100, 200, 500, 600],
```

```
#    'xgb__max_depth': [3, 5, 7, 10, 15, 20, 25],
#    'xgb__subsample': [0.5, 0.7, 1.0],
#    'xgb__colsample_bytree': [0.5, 0.7, 1.0]
#}

#model_xgb, params_xgb, _, _ = train_crossval_predict_score(
#    xgb_pipe,
#    hyperparameters,
#    X_train,
#    y_train,
#    X_test,
#    y_test,
#    cv=5,
#    scoring='f1',
#    verbose=0,
#    n_jobs=-1,
#    cross_val='full',
#    random_state=None,
#    training=True,
#    test=True,
#    accuracy=True,
#    recall=True,
#    precision=True,
#    fbeta=[True, 1.0],
#    roc_auc=True,
#    matrix=True,
#    figsize=(3,2),
#    cmap='YlGn')
```

Full Grid search: Best params: {'xgb___colsample_bytree': 1.0, 'xgb___learning_rate': 0.01, 'xgb___max_depth': 7, 'xgb___n_estimators': 600, 'xgb___subsample': 0.7}

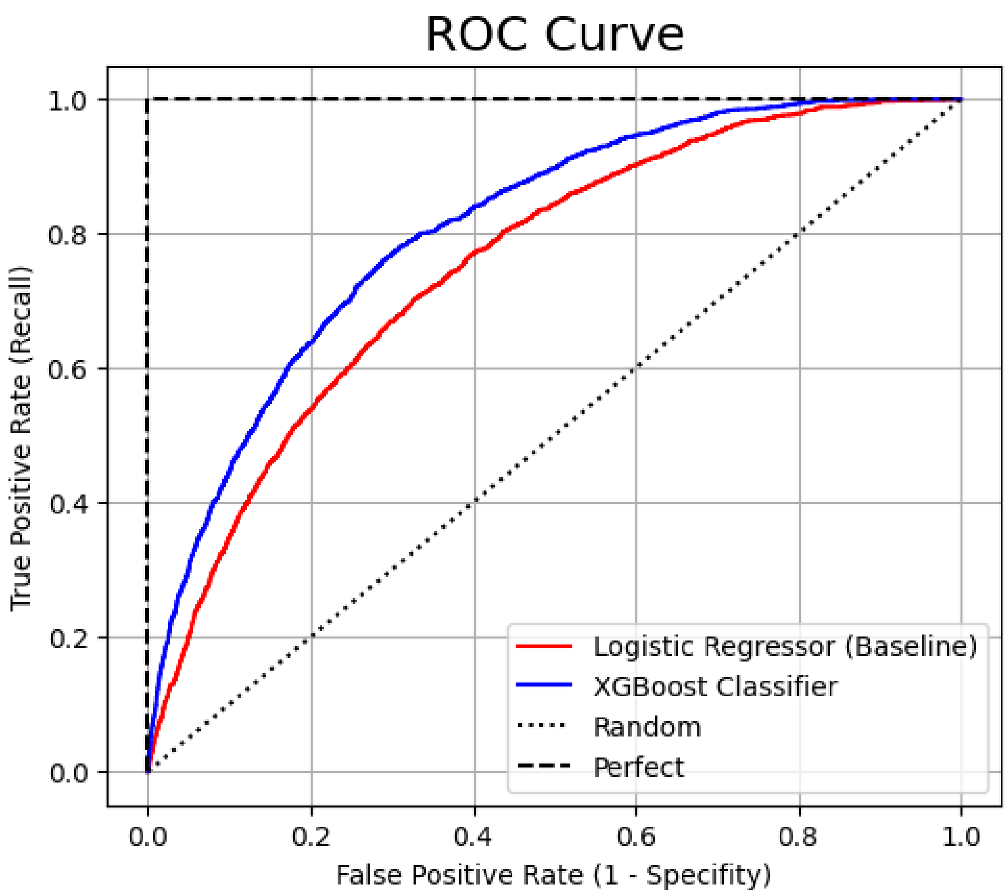
[59]:
```
# Separate features by type
num_features = X_train.select_dtypes(include=['int32', 'float64']).columns
cat_features = X_train.select_dtypes(include=['object', 'category']).columns

# Preprocessing for numerical data
num_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value=0))
])

# Preprocessing for categorical data
cat_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value=0)),
])

# Apply preprocessing to num and categorical features
```

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', num_transformer, num_features),
        ('cat', cat_transformer, cat_features)
    ],
    remainder='passthrough')

model_xgb = ImbPipeline([
    ('prep', preprocessor),
    ('undersample', RandomUnderSampler(sampling_strategy=0.5)),
    ('best_xgb', XGBClassifier(n_estimators=600, max_depth=7, learning_rate=0.
 ↪01, colsample_bytree=1.0, subsample=0.7, use_label_encoder=False))
])

# Fit model
model_xgb.fit(X_train, y_train)

# Predict and score
predict_and_print_scores(model_xgb, X_train, y_train, X_test, y_test,␣
 ↪matrix=False)
```

```
Accuracy on training set: 0.87
Accuracy on test set: 0.86
----------------------------------------
Recall on training set: 0.65
Recall on test set: 0.49
----------------------------------------
Precision on training set: 0.25
Precision on test set: 0.19
----------------------------------------
fbeta_score on training set: 0.36
fbeta_score on test set: 0.28
----------------------------------------
roc_auc_score on training set:  0.88
roc_auc_score on test set:  0.81
----------------------------------------
```

11 Model Evaluation

[65]:
```
models = {'Logistic Regressor (Baseline)' : [model_lg, 'r-'], 'XGBoost␣
 ↪Classifier' : [model_xgb, 'b-']}
fig, ax = plot_roc_curves(models, X_test, y_test)
```



As observed, the **XGBoost** classifier outperforms the baseline model, but requires a total of 600 week estimators to make better predictions.

[61]:
```
# Find the threshold that maximizes f1 for final prediction
best_thr, score = find_roc_threshold_f1(model_xgb, X_test, y_test)
print(f"Best threshold: {round(best_thr, 2)}")
print(f"Best f1 score: {round(score, 2)}")
```

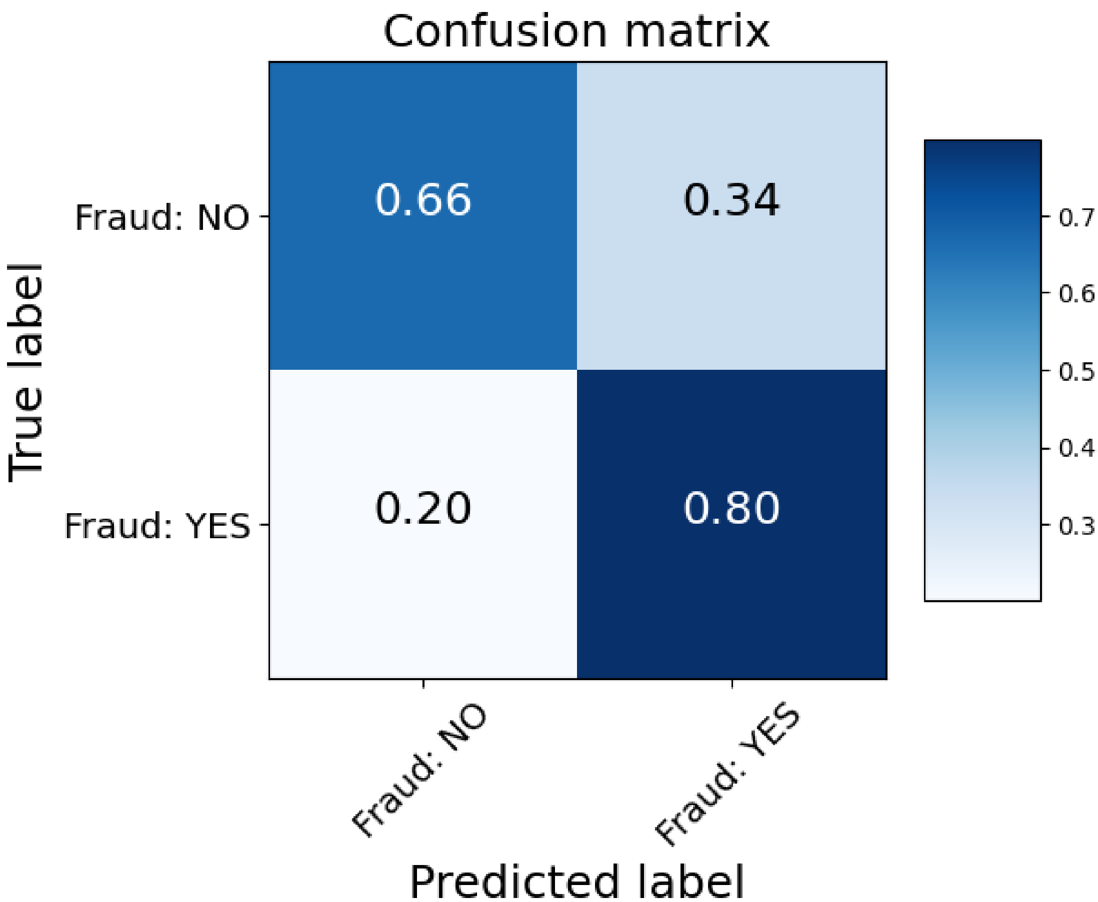
```
Best threshold: 0.6
Best f1 score: 0.29
```

[70]:
```
# Find the false positive rate corresponding to a recall of 80%
thr_80, false_pos_rate = find_roc_threshold_tpr(model_xgb, X_test, y_test, 0.8)

print(f"Threshold for a recall of 80%: {round(thr_80, 2)}")
print(f"False positive rate: {round(false_pos_rate, 2)}")
```

```
Threshold for a recall of 80%: 0.3199999928474426
False positive rate: 0.34
```

[69]:
```
# Plot confusion matrix on the test set
y_test_pred = model_xgb.predict_proba(X_test)[:,1] >= thr_80
cm = confusion_matrix(y_test, y_test_pred)
plot_confusion_matrix(cm, ['Fraud: NO', 'Fraud: YES'], normalize=True,␣
 ↪title='Confusion matrix', cmap=plt.cm.Blues, figsize=(6,6))
```

```
Normalized confusion matrix
```



The XGBoost classifier is able to detect with a **recall rate of 80%** and a **false positive rate of 34%**.

[66]:
```
# Find out the most relevant features

# Get the XGB Classifier step from the pipeline
xgb_step = model_xgb.named_steps['best_xgb']

# Access feature importance
feature_importance = xgb_step.feature_importances_
feature_names = X_train.columns
```

```python
# Combine feature names and their importance into a dictionary, then sort by␣
↪importance
importance_dict = dict(zip(feature_names, feature_importance))
sorted_importance = sorted(importance_dict.items(), key=lambda x: x[1],␣
↪reverse=True)

# Displaying feature importance
for feature, importance in sorted_importance:
    print(f"Feature: {feature}, Importance: {round(100 * importance, 2)}%")
```

```
Feature: cons3_std, Importance: 11.12%
Feature: transaction_count, Importance: 9.21%
Feature: cons3_min, Importance: 3.75%
Feature: cons4_std, Importance: 2.72%
Feature: cons2_std, Importance: 2.71%
Feature: cons2_range, Importance: 2.68%
Feature: ener_total_consumption, Importance: 2.56%
Feature: counter_code_mean, Importance: 2.34%
Feature: invoice_per_cooperation, Importance: 2.33%
Feature: counter_code_std, Importance: 2.25%
Feature: counter_status_min, Importance: 2.09%
Feature: cons4_min, Importance: 1.98%
Feature: counter_code_max, Importance: 1.97%
Feature: counter_coefficient_std, Importance: 1.97%
Feature: counter_code_min, Importance: 1.95%
Feature: cons1_max, Importance: 1.93%
Feature: cons3_range, Importance: 1.81%
Feature: cons3_mean, Importance: 1.8%
Feature: counter_status_mean, Importance: 1.78%
Feature: cons3_max, Importance: 1.74%
Feature: cons2_mean, Importance: 1.73%
Feature: agent_remark_max, Importance: 1.73%
Feature: cons2_min, Importance: 1.69%
Feature: coop_time, Importance: 1.65%
Feature: cons1_total, Importance: 1.65%
Feature: cons2_max, Importance: 1.62%
Feature: cons1_std, Importance: 1.62%
Feature: agent_remark_mean, Importance: 1.6%
Feature: client_catg, Importance: 1.57%
Feature: cons4_range, Importance: 1.55%
Feature: ener_min_invoice_delta, Importance: 1.55%
Feature: ener_std_invoice_delta, Importance: 1.54%
Feature: cons1_range, Importance: 1.51%
Feature: cons4_max, Importance: 1.5%
Feature: cons4_mean, Importance: 1.45%
Feature: ener_min_consumption, Importance: 1.39%
```

```
Feature: ener_mean_consumption, Importance: 1.35%
Feature: ener_max_consumption, Importance: 1.34%
Feature: cons1_mean, Importance: 1.33%
Feature: ener_std_consumption, Importance: 1.29%
Feature: counter_status_max, Importance: 1.26%
Feature: ener_max_invoice_delta, Importance: 1.24%
Feature: cons1_min, Importance: 1.18%
Feature: ener_mean_invoice_delta, Importance: 1.13%
Feature: cons3_total, Importance: 1.03%
Feature: counter_status_std, Importance: 0.72%
Feature: cons2_total, Importance: 0.71%
Feature: agent_remark_min, Importance: 0.57%
Feature: cons4_total, Importance: 0.49%
Feature: region, Importance: 0.31%
Feature: agent_remark_std, Importance: 0.0%
Feature: counter_coefficient_min, Importance: 0.0%
Feature: counter_coefficient_max, Importance: 0.0%
Feature: counter_coefficient_mean, Importance: 0.0%
```

As observed, the partial and total consumption levels, the number of invoices, the counter information (coefficient, code, status), as well as the agent remarks seem to be the most relevant features to detect fraud.

12 Conclusions

The detection of fraudulent cases with a moderate degree of accuracy has required a lot of feature engineering, since the features in the dataset were not good enough to be able to detect such cases. The target output was also imbalanced which also affected the overall results.

In general, boosting methods provided, for this type of problems where data quality is not good, better results than other approaches such as Random Forest. Microsoft's LGBMBoost and AdaBoost (not included in the notebook for the sake of simplicity) resulted in slightly lower performance than XGBoost.