!pip install catboost

Collecting catboost

anylinux1\_x86\_64.whl (66.1MB)

ent already satisfied: graphviz in /usr/local/lib/python3.6/dist-packages (from catboost) (0.10.1)

Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages (from catboost) (3.2.2)

Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from catboost) (1.15.0)

Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (from catboost) (4.4.1)

Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from catboost) (1.4.1)

Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.6/dist-packages (from catboost) (1.1.2)

Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.6/dist-packages (from catboost) (1.18.5)

Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->catboost) (2.4.7)

Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib->catboost) (0.10.0)

Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->catboost) (1.2.0)

Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->catboost) (2.8.1)

Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dist-packages (from plotly->catboost) (1.3.3)

Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.24.0->catboost) (2018.9)

Installing collected packages: catboost

Successfully installed catboost-0.24.2

# Импорт библиотек

import pandas as pd

import numpy as np

import matplotlib as mpl

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.model\_selection import train\_test\_split, cross\_validate, GridSearchCV, cross\_val\_score

from imblearn.under\_sampling import RandomUnderSampler

from sklearn.metrics import accuracy\_score, f1\_score, confusion\_matrix, classification\_report

from sklearn.pipeline import make\_pipeline

from sklearn.model\_selection import StratifiedKFold

from xgboost import XGBClassifier

from lightgbm import LGBMClassifier

from catboost import CatBoostClassifier

# задаем стиль графических построений

sns.set\_style()

# скрываем предупреждающие сообщения

import warnings

warnings.filterwarnings('ignore')

/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31: FutureWarning: The module is deprecated in version 0.21 and will be removed in version 0.23 since we've dropped support for Python 2.7. Please rely on the official version of six (https://pypi.org/project/six/).

"(https://pypi.org/project/six/).", FutureWarning)

/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning: The sklearn.neighbors.base module is deprecated in version 0.22 and will be removed in version 0.24. The corresponding classes / functions should instead be imported from sklearn.neighbors. Anything that cannot be imported from sklearn.neighbors is now part of the private API.

warnings.warn(message, FutureWarning)

# установливаем параметры matplotlib по умолчанию

COLOR = '#ababab'

mpl.rcParams['figure.titlesize'] = 16

mpl.rcParams['text.color'] = 'black'

mpl.rcParams['axes.labelcolor'] = COLOR

mpl.rcParams['xtick.color'] = COLOR

mpl.rcParams['ytick.color'] = COLOR

mpl.rcParams['grid.color'] = COLOR

mpl.rcParams['grid.alpha'] = 0.1

# импортируем набор данных и создайте датафрейм

df\_credit = pd.read\_csv('http://dl.dropboxusercontent.com/s/xn2a4kzf0zer0xu/acquisition\_train.csv?dl=0')

# Просмотр визуально данных датафрейма

df\_credit.head()

ids ... target\_fraud

0 343b7e7b-2cf8-e508-b8fd-0a0285af30aa ... NaN

1 bc2c7502-bbad-0f8c-39c3-94e881967124 ... NaN

2 669630dd-2e6a-0396-84bf-455e5009c922 ... NaN

3 d235609e-b6cb-0ccc-a329-d4f12e7ebdc1 ... NaN

4 9e0eb880-e8f4-3faa-67d8-f5cdd2b3932b ... NaN

[5 rows x 43 columns]

# Выводим сведения о количестве строк и число полей (атрибутов)

print('Number of rows: ', df\_credit.shape[0])

print('Number of columns: ', df\_credit.shape[1])

Number of rows: 45000

Number of columns: 43

# Выводим сведения о количестве null полей (атрибутов)

df\_credit.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 45000 entries, 0 to 44999

Data columns (total 43 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 ids 45000 non-null object

1 target\_default 41741 non-null object

2 score\_1 44438 non-null object

3 score\_2 44438 non-null object

4 score\_3 44438 non-null float64

5 score\_4 45000 non-null float64

6 score\_5 45000 non-null float64

7 score\_6 45000 non-null float64

8 risk\_rate 44438 non-null float64

9 last\_amount\_borrowed 15044 non-null float64

10 last\_borrowed\_in\_months 15044 non-null float64

11 credit\_limit 31200 non-null float64

12 reason 44434 non-null object

13 income 44438 non-null float64

14 facebook\_profile 40542 non-null object

15 state 44438 non-null object

16 zip 44438 non-null object

17 channel 44438 non-null object

18 job\_name 41664 non-null object

19 real\_state 44438 non-null object

20 ok\_since 18455 non-null float64

21 n\_bankruptcies 44303 non-null float64

22 n\_defaulted\_loans 44426 non-null float64

23 n\_accounts 44438 non-null float64

24 n\_issues 33456 non-null float64

25 application\_time\_applied 45000 non-null object

26 application\_time\_in\_funnel 45000 non-null int64

27 email 45000 non-null object

28 external\_data\_provider\_credit\_checks\_last\_2\_year 22372 non-null float64

29 external\_data\_provider\_credit\_checks\_last\_month 45000 non-null int64

30 external\_data\_provider\_credit\_checks\_last\_year 29876 non-null float64

31 external\_data\_provider\_email\_seen\_before 42767 non-null float64

32 external\_data\_provider\_first\_name 45000 non-null object

33 external\_data\_provider\_fraud\_score 45000 non-null int64

34 lat\_lon 43637 non-null object

35 marketing\_channel 41422 non-null object

36 profile\_phone\_number 45000 non-null object

37 reported\_income 45000 non-null float64

38 shipping\_state 45000 non-null object

39 shipping\_zip\_code 45000 non-null int64

40 profile\_tags 45000 non-null object

41 user\_agent 44278 non-null object

42 target\_fraud 1522 non-null object

dtypes: float64(18), int64(4), object(21)

memory usage: 14.8+ MB

Значения некоторых полей незаполнены

# процент пропущенных значений для каждого поля (атрибута)

print((df\_credit.isnull().sum() \* 100 / df\_credit.shape[0]).sort\_values(ascending=False))

target\_fraud 96.617778

last\_amount\_borrowed 66.568889

last\_borrowed\_in\_months 66.568889

ok\_since 58.988889

external\_data\_provider\_credit\_checks\_last\_2\_year 50.284444

external\_data\_provider\_credit\_checks\_last\_year 33.608889

credit\_limit 30.666667

n\_issues 25.653333

facebook\_profile 9.906667

marketing\_channel 7.951111

job\_name 7.413333

target\_default 7.242222

external\_data\_provider\_email\_seen\_before 4.962222

lat\_lon 3.028889

user\_agent 1.604444

n\_bankruptcies 1.548889

n\_defaulted\_loans 1.275556

reason 1.257778

income 1.248889

real\_state 1.248889

state 1.248889

zip 1.248889

channel 1.248889

score\_3 1.248889

score\_2 1.248889

score\_1 1.248889

n\_accounts 1.248889

risk\_rate 1.248889

shipping\_zip\_code 0.000000

score\_4 0.000000

score\_5 0.000000

profile\_tags 0.000000

score\_6 0.000000

application\_time\_in\_funnel 0.000000

shipping\_state 0.000000

reported\_income 0.000000

application\_time\_applied 0.000000

profile\_phone\_number 0.000000

external\_data\_provider\_fraud\_score 0.000000

external\_data\_provider\_first\_name 0.000000

external\_data\_provider\_credit\_checks\_last\_month 0.000000

email 0.000000

ids 0.000000

dtype: float64

# удалим все записи, где target\_default (наша целевая переменная) равно null.

df\_credit.dropna(subset=['target\_default'], inplace=True)

# Удалим колонку с атрибутом "target\_fraud", в которй отсутствуют почти все записи

df\_credit.drop('target\_fraud', axis=1, inplace=True)

# количество уникальных значений для каждого атрибута

df\_credit.nunique().sort\_values()

channel 1

external\_data\_provider\_credit\_checks\_last\_2\_year 1

last\_borrowed\_in\_months 2

target\_default 2

facebook\_profile 2

external\_data\_provider\_credit\_checks\_last\_year 2

external\_data\_provider\_credit\_checks\_last\_month 4

real\_state 5

n\_defaulted\_loans 5

email 6

n\_bankruptcies 6

score\_1 7

marketing\_channel 9

shipping\_state 25

score\_2 35

n\_issues 44

n\_accounts 44

state 50

external\_data\_provider\_email\_seen\_before 62

risk\_rate 81

score\_3 87

ok\_since 100

user\_agent 297

application\_time\_in\_funnel 501

zip 823

external\_data\_provider\_fraud\_score 1001

last\_amount\_borrowed 13480

reason 14260

credit\_limit 19336

lat\_lon 21596

profile\_tags 24458

shipping\_zip\_code 26996

job\_name 30543

external\_data\_provider\_first\_name 31183

application\_time\_applied 33560

reported\_income 37368

income 38849

score\_5 41741

profile\_phone\_number 41741

score\_4 41741

score\_6 41741

ids 41741

dtype: int64

# Удалим колонки "channel" и "external\_data\_provider\_credit\_checks\_last\_2\_year"

df\_credit.drop(labels=['channel', 'external\_data\_provider\_credit\_checks\_last\_2\_year'], axis=1, inplace=True)

# удалим некоторые столбцы, которые не добавляют ценности модели

df\_credit.drop(labels=['email', 'reason', 'zip', 'job\_name', 'external\_data\_provider\_first\_name', 'lat\_lon', 'shipping\_zip\_code', 'user\_agent', 'profile\_tags', 'marketing\_channel', 'profile\_phone\_number', 'application\_time\_applied', 'ids'], axis=1, inplace=True)

# проверим, есть ли в наборе данных выбросы. Покажем статистику

df\_credit.describe()

score\_3 ... reported\_income

count 41741.000000 ... 41741.0

mean 346.459836 ... inf

std 110.102271 ... NaN

min 0.000000 ... 403.0

25% 270.000000 ... 50910.0

50% 340.000000 ... 101623.0

75% 420.000000 ... 151248.0

max 990.000000 ... inf

[8 rows x 20 columns]

# количество значений "inf" в "reported\_income"

np.isinf(df\_credit['reported\_income']).sum()

66

# количество значений = -999 в "external\_data\_provider\_email\_seen\_before"

df\_credit.loc[df\_credit['external\_data\_provider\_email\_seen\_before'] == -999, 'external\_data\_provider\_email\_seen\_before'].value\_counts()

-999.0 591

Name: external\_data\_provider\_email\_seen\_before, dtype: int64

заменим эти значения "inf" и "-999" на NaN, чтобы мы могли построить гистограммы для визуализации распределения значений

# замена "inf" на "nan"

df\_credit['reported\_income'] = df\_credit['reported\_income'].replace(np.inf, np.nan)

# замена "-999" на "nan"

df\_credit.loc[df\_credit['external\_data\_provider\_email\_seen\_before'] == -999, 'external\_data\_provider\_email\_seen\_before'] = np.nan

Построим гистограммы для этих характеристик, что поможет изучить их распределение

# фрейм данных, содержащий числовые характеристики

df\_credit\_numerical = df\_credit[['score\_3', 'risk\_rate', 'last\_amount\_borrowed','last\_borrowed\_in\_months', 'credit\_limit', 'income', 'ok\_since','n\_bankruptcies', 'n\_defaulted\_loans', 'n\_accounts', 'n\_issues', 'external\_data\_provider\_email\_seen\_before']]

# построим гистограммы для каждого из приведенных выше объектов

nrows = 3

ncols = 4

fig, ax = plt.subplots(nrows=nrows, ncols=ncols, figsize=(25, 16))

r = 0

c = 0

for i in df\_credit\_numerical:

sns.distplot(df\_credit\_numerical[i], bins=15,kde=False, ax=ax[r][c])

if c == ncols - 1:

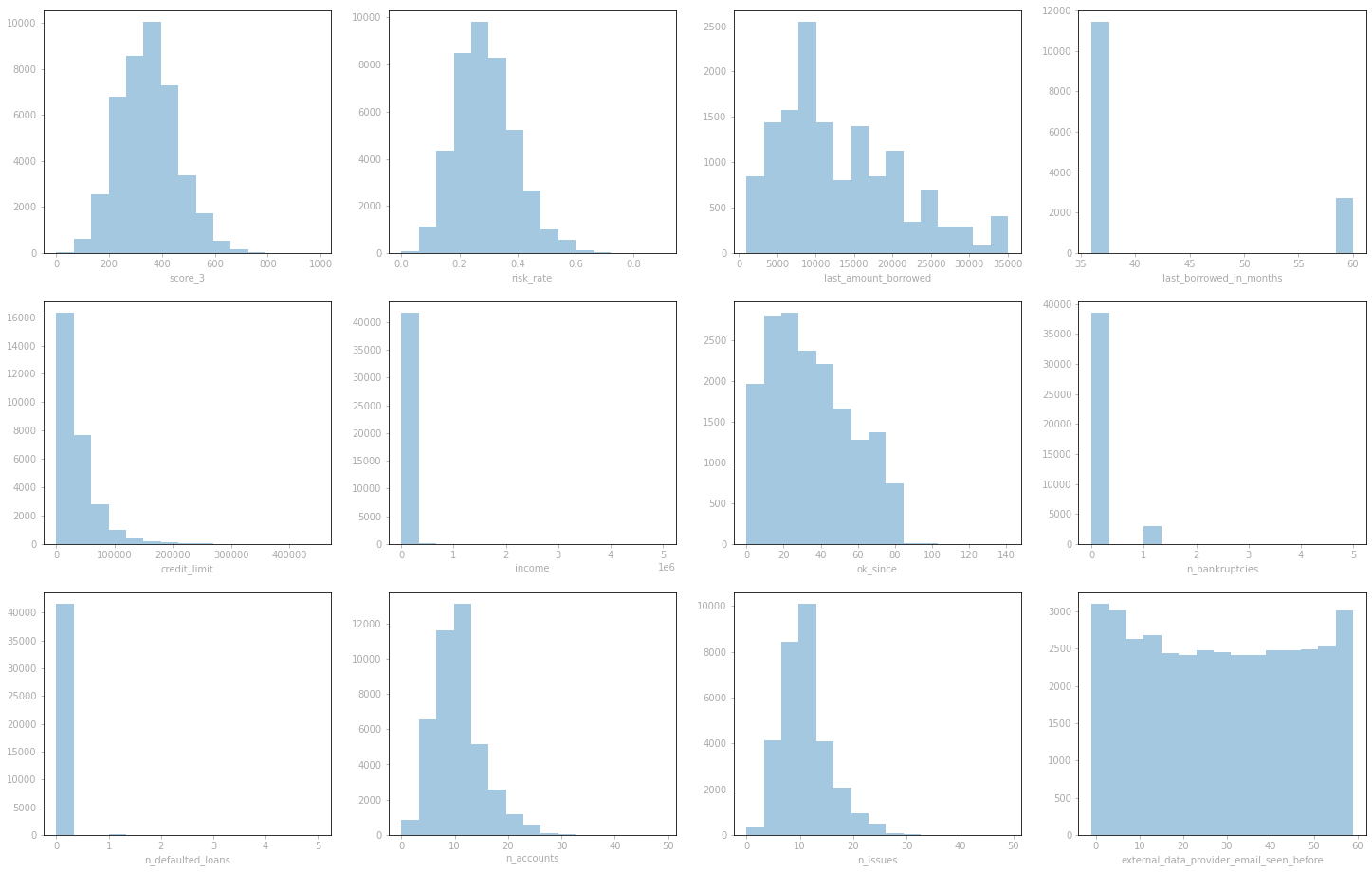
r += 1

c = 0

else:

c += 1

plt.show()



Все перечисленные выше объекты имеют пропущенные значения, которые необходимо обработать. Как мы можем видеть, они имеют искаженное распределение, что указывает на то, что мы должны заполнить пропущенные значения медианным значением для каждого объекта.

Заполняем значения из оставшихся 32 столбцов с пропущенными значениями в соответствии с особенностями каждого признака, как показано ниже:

* Категориальные переменные будут заполнены наиболее повторяющимся значением.
* Числовые переменные будут заполнены их средними значениями.
* В конкретных случаях last\_amount\_borrowed, last\_borrowed\_in\_months и n\_issues мы заполним недостающие значения нулем, поскольку разумно полагать, что не каждому клиенту будут присвоены значения этим переменным.

df\_credit\_num = df\_credit.select\_dtypes(exclude='object').columns

df\_credit\_cat = df\_credit.select\_dtypes(include='object').columns

# fill missing values for "last\_amount\_borrowed", "last\_borrowed\_in\_months" and "n\_issues"

df\_credit['last\_amount\_borrowed'].fillna(value=0, inplace=True)

df\_credit['last\_borrowed\_in\_months'].fillna(value=0, inplace=True)

df\_credit['n\_issues'].fillna(value=0, inplace=True)

# fill missing values for numerical variables

imputer = SimpleImputer(missing\_values=np.nan, strategy='median')

imputer = imputer.fit(df\_credit.loc[:, df\_credit\_num])

df\_credit.loc[:, df\_credit\_num] = imputer.transform(df\_credit.loc[:, df\_credit\_num])

# fill missing values for categorical variables

imputer = SimpleImputer(missing\_values=np.nan, strategy='most\_frequent')

imputer = imputer.fit(df\_credit.loc[:, df\_credit\_cat])

df\_credit.loc[:, df\_credit\_cat] = imputer.transform(df\_credit.loc[:, df\_credit\_cat])

# Проверяем полученный новый датафрейм на наличие нулейвых значений полей (атрибутов)

df\_credit.isnull().sum()

target\_default 0

score\_1 0

score\_2 0

score\_3 0

score\_4 0

score\_5 0

score\_6 0

risk\_rate 0

last\_amount\_borrowed 0

last\_borrowed\_in\_months 0

credit\_limit 0

income 0

facebook\_profile 0

state 0

real\_state 0

ok\_since 0

n\_bankruptcies 0

n\_defaulted\_loans 0

n\_accounts 0

n\_issues 0

application\_time\_in\_funnel 0

external\_data\_provider\_credit\_checks\_last\_month 0

external\_data\_provider\_credit\_checks\_last\_year 0

external\_data\_provider\_email\_seen\_before 0

external\_data\_provider\_fraud\_score 0

reported\_income 0

shipping\_state 0

dtype: int64

Проведем предварительную обработку данных, преобразовав категориальные признаки в числовые значения. LabelEncoder будет использоваться для двоичных переменных, в то время как get\_dummies будет использоваться для других категориальных переменных.

bin\_var = df\_credit.nunique()[df\_credit.nunique() == 2].keys().tolist()

num\_var = [col for col in df\_credit.select\_dtypes(['int', 'float']).columns.tolist() if col not in bin\_var]

cat\_var = [col for col in df\_credit.select\_dtypes(['object']).columns.tolist() if col not in bin\_var]

df\_credit\_encoded = df\_credit.copy()

# label encoding for the binary variables

le = LabelEncoder()

for col in bin\_var:

df\_credit\_encoded[col] = le.fit\_transform(df\_credit\_encoded[col])

# encoding with get\_dummies for the categorical variables

df\_credit\_encoded = pd.get\_dummies(df\_credit\_encoded, columns=cat\_var)

df\_credit\_encoded.head()

target\_default score\_3 ... shipping\_state\_BR-SP shipping\_state\_BR-TO

0 0 350.0 ... 0 0

1 0 370.0 ... 0 0

2 1 360.0 ... 0 0

3 0 510.0 ... 0 0

4 0 500.0 ... 0 0

[5 rows x 144 columns]

Работа над моделями машинного обучения.

# Feature matrix — это инструмент управления продуктом, который помогает командам определить, какие функции разрабатывать дальше.

X = df\_credit\_encoded.drop('target\_default', axis=1)

# целевая переменная (целевой атрибут)

y = df\_credit\_encoded['target\_default']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, shuffle=True, stratify=y)

Поскольку мы имеем дело с несбалансированным набором данных, мы стандартизируем и повторно выполним выборку обучающего набора с помощью StandardScaler и RandomUnderSampler соответственно.

# стандартизируем числовые переменные

scaler = StandardScaler().fit(X\_train)

X\_train = scaler.transform(X\_train)

# повторная выборка

rus = RandomUnderSampler()

X\_train\_rus, y\_train\_rus = rus.fit\_sample(X\_train, y\_train)

# определим функцию val\_model

def val\_model(X, y, clf, show=True):

"""

Apply cross-validation on the training set.

# Arguments X: DataFrame containing the independent variables.

y: Series containing the target vector.

clf: Scikit-learn estimator instance.

# Returns float, mean value of the cross-validation scores.

"""

X = np.array(X)

y = np.array(y)

pipeline = make\_pipeline(StandardScaler(), clf)

scores = cross\_val\_score(pipeline, X, y, scoring='recall')

if show == True:

print(f'Recall: {scores.mean()}, {scores.std()}')

return scores.mean()

#оценим модели

xgb = XGBClassifier()

lgb = LGBMClassifier()

cb = CatBoostClassifier()

model = []

recall = []

for clf in (xgb, lgb, cb):

model.append(clf.\_\_class\_\_.\_\_name\_\_)

recall.append(val\_model(X\_train\_rus, y\_train\_rus, clf, show=False))

pd.DataFrame(data=recall, index=model, columns=['Recall'])

# XGBoost

xgb = XGBClassifier()

# параметр, подлежащий поиску

param\_grid = {'n\_estimators': range(0,1000,50)}

# найдем наилучший параметр

kfold = StratifiedKFold(n\_splits=3, shuffle=True)

grid\_search = GridSearchCV(xgb, param\_grid, scoring="recall", n\_jobs=-1, cv=kfold)

grid\_result = grid\_search.fit(X\_train\_rus, y\_train\_rus)

print(f'Best result: {grid\_result.best\_score\_} for {grid\_result.best\_params\_}')

Best result: 0.6657327195142321 for {'n\_estimators': 50}

# XGBoost

xgb = XGBClassifier(n\_estimators=50)

# параметр, подлежащий поиску

param\_grid = {'max\_depth': [1, 3, 5], 'min\_child\_weight': [1, 3, 6]}

# найдем наилучший параметр

kfold = StratifiedKFold(n\_splits=3, shuffle=True)

grid\_search = GridSearchCV(xgb, param\_grid, scoring="recall", n\_jobs=-1, cv=kfold)

grid\_result = grid\_search.fit(X\_train\_rus, y\_train\_rus)

print(f'Best result: {grid\_result.best\_score\_} for {grid\_result.best\_params\_}')

Best result: 0.6701307550047045 for {'max\_depth': 3, 'min\_child\_weight': 6}

# XGBoost

xgb = XGBClassifier(n\_estimators=50, max\_depth=3, min\_child\_weight=6)

# параметр, подлежащий поиску

param\_grid = {'gamma': [0, 1, 5]}

# найдем наилучший параметр

kfold = StratifiedKFold(n\_splits=3, shuffle=True)

grid\_search = GridSearchCV(xgb, param\_grid, scoring="recall", n\_jobs=-1, cv=kfold)

grid\_result = grid\_search.fit(X\_train\_rus, y\_train\_rus)

print(f'Best result: {grid\_result.best\_score\_} for {grid\_result.best\_params\_}')

Best result: 0.6719385652158761 for {'gamma': 1}

# XGBoost

xgb = XGBClassifier(n\_estimators=50, max\_depth=3, min\_child\_weight=6, gamma=1)

# параметр, подлежащий поиску

param\_grid = {'learning\_rate': [0.0001, 0.001, 0.01, 0.1]}

# найдем наилучший параметр

kfold = StratifiedKFold(n\_splits=3, shuffle=True)

grid\_search = GridSearchCV(xgb, param\_grid, scoring='recall', n\_jobs=-1, cv=kfold)

grid\_result = grid\_search.fit(X\_train\_rus, y\_train\_rus)

print(f'Best result: {grid\_result.best\_score\_} for {grid\_result.best\_params\_}')

Best result: 0.8170548699960465 for {'learning\_rate': 0.0001}

# LightGBM

lbg = LGBMClassifier(silent=False)

# параметр, подлежащий поиску

param\_grid = {"max\_depth": np.arange(5, 75, 10),

"learning\_rate" : [0.001, 0.01, 0.1],

"num\_leaves": np.arange(20, 220, 50),

}

# найдем наилучший параметр

kfold = StratifiedKFold(n\_splits=3, shuffle=True)

grid\_search = GridSearchCV(lbg, param\_grid, scoring="recall", n\_jobs=-1, cv=kfold)

grid\_result = grid\_search.fit(X\_train\_rus, y\_train\_rus)

print(f'Best result: {grid\_result.best\_score\_} for {grid\_result.best\_params\_}')

Best result: 0.6883483723819858 for {'learning\_rate': 0.01, 'max\_depth': 5, 'num\_leaves': 70}

lbg = LGBMClassifier(learning\_rate=0.01, max\_depth=5, num\_leaves=50, silent=False)

# параметр, подлежащий поиску

param\_grid = {'min\_data\_in\_leaf': np.arange(100, 1000, 100)}

# найдем наилучший параметр

kfold = StratifiedKFold(n\_splits=3, shuffle=True)

grid\_search = GridSearchCV(lbg, param\_grid, scoring="recall", n\_jobs=-1, cv=kfold)

grid\_result = grid\_search.fit(X\_train\_rus, y\_train\_rus)

print(f'Best result: {grid\_result.best\_score\_} for {grid\_result.best\_params\_}')

Best result: 0.6961582230489793 for {'min\_data\_in\_leaf': 400}

# CatBoost

cb = CatBoostClassifier()

# параметр, подлежащий поиску

param\_grid = {'depth': [6, 8, 10],

'learning\_rate': [0.03, 0.1],

'l2\_leaf\_reg': [1, 5, 10],

}

# найдем наилучший параметрkfold = StratifiedKFold(n\_splits=3, shuffle=True)

grid\_search = GridSearchCV(cb, param\_grid, scoring="recall", n\_jobs=-1, cv=kfold)

grid\_result = grid\_search.fit(X\_train\_rus, y\_train\_rus)

print(f'Best result: {grid\_result.best\_score\_} for {grid\_result.best\_params\_}')

0: learn: 0.6895401 total: 9.95ms remaining: 9.94s

1: learn: 0.6860938 total: 18.7ms remaining: 9.31s

2: learn: 0.6829368 total: 26.7ms remaining: 8.88s

3: learn: 0.6798871 total: 34.4ms remaining: 8.57s

4: learn: 0.6772491 total: 42.5ms remaining: 8.45s

5: learn: 0.6746800 total: 50.1ms remaining: 8.31s

6: learn: 0.6724516 total: 58ms remaining: 8.23s

…

995: learn: 0.4568061 total: 8.17s remaining: 32.8ms

996: learn: 0.4566922 total: 8.18s remaining: 24.6ms

997: learn: 0.4565555 total: 8.19s remaining: 16.4ms

998: learn: 0.4564079 total: 8.19s remaining: 8.2ms

999: learn: 0.4563245 total: 8.2s remaining: 0us

Best result: 0.6561276762957435 for {'depth': 6, 'l2\_leaf\_reg': 5, 'learning\_rate': 0.03}

Матрица неточностей (англ. Confusion Matrix) — это таблица или диаграмма, показывающая точность прогнозирования классификатора в отношении двух и более классов. Прогнозы классификатора находятся на оси X, а результат (точность) — на оси Y.

# финальная модель XGBoost

xgb = XGBClassifier(max\_depth=3, learning\_rate=0.0001, n\_estimators=50, gamma=1, min\_child\_weight=6)

xgb.fit(X\_train\_rus, y\_train\_rus)

# Прогноз модели (Prediction)

X\_test\_xgb = scaler.transform(X\_test)

y\_pred\_xgb = xgb.predict(X\_test\_xgb)

# вывод на экран классификации

print(classification\_report(y\_test, y\_pred\_xgb))

# confusion matrix

fig, ax = plt.subplots()

sns.heatmap(confusion\_matrix(y\_test, y\_pred\_xgb, normalize='true'), annot=True, ax=ax)

ax.set\_title('Confusion Matrix - XGBoost')

ax.set\_xlabel('Predicted Value')

ax.set\_ylabel('Real Value')

plt.show()

precision recall f1-score support

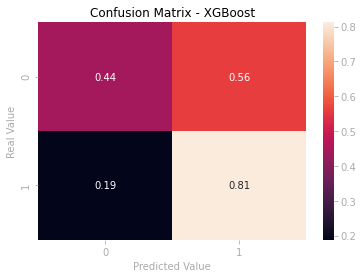
0 0.92 0.44 0.60 8771

1 0.22 0.81 0.34 1665

accuracy 0.50 10436

macro avg 0.57 0.63 0.47 10436

weighted avg 0.81 0.50 0.56 10436



# final LightGBM model

lgb = LGBMClassifier(num\_leaves=70, max\_depth=5, learning\_rate=0.01, min\_data\_in\_leaf=400)

lgb.fit(X\_train\_rus, y\_train\_rus)

# Прогноз модели (Prediction)

X\_test\_lgb = scaler.transform(X\_test)

y\_pred\_lgb = lgb.predict(X\_test\_lgb)

# вывод на экран классификации

print(classification\_report(y\_test, y\_pred\_lgb))

# confusion matrix

fig, ax = plt.subplots()

sns.heatmap(confusion\_matrix(y\_test, y\_pred\_lgb, normalize='true'), annot=True, ax=ax)

ax.set\_title('Confusion Matrix - LightGBM')

ax.set\_xlabel('Predicted Value')

ax.set\_ylabel('Real Value')

plt.show()

precision recall f1-score support

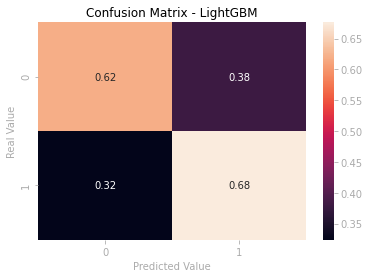
0 0.91 0.62 0.74 8771

1 0.25 0.68 0.37 1665

accuracy 0.63 10436

macro avg 0.58 0.65 0.55 10436

weighted avg 0.81 0.63 0.68 10436



# final CatBoost model

cb = CatBoostClassifier(learning\_rate=0.03, depth=6, l2\_leaf\_reg=5, logging\_level='Silent')

cb.fit(X\_train\_rus, y\_train\_rus)

# Прогноз модели (Prediction)

X\_test\_cb = scaler.transform(X\_test)

y\_pred\_cb = cb.predict(X\_test\_cb)

# Сведения о классификации

print(classification\_report(y\_test, y\_pred\_cb))

# Матрица неточностей (англ. Confusion Matrix) — это таблица или диаграмма,

# показывающая точность прогнозирования классификатора в отношении двух и более классов.

# Прогнозы классификатора находятся на оси X, а результат (точность) — на оси Y.

fig, ax = plt.subplots()

sns.heatmap(confusion\_matrix(y\_test, y\_pred\_cb, normalize='true'), annot=True, ax=ax)

ax.set\_title('Confusion Matrix - CatBoost')

ax.set\_xlabel('Predicted Value')

ax.set\_ylabel('Real Value')

plt.show()

precision recall f1-score support

0 0.91 0.67 0.77 8771

1 0.27 0.64 0.38 1665

accuracy 0.66 10436

macro avg 0.59 0.65 0.57 10436

weighted avg 0.81 0.66 0.71 10436

