Урок 4.

Способы улучшения полученной модели.

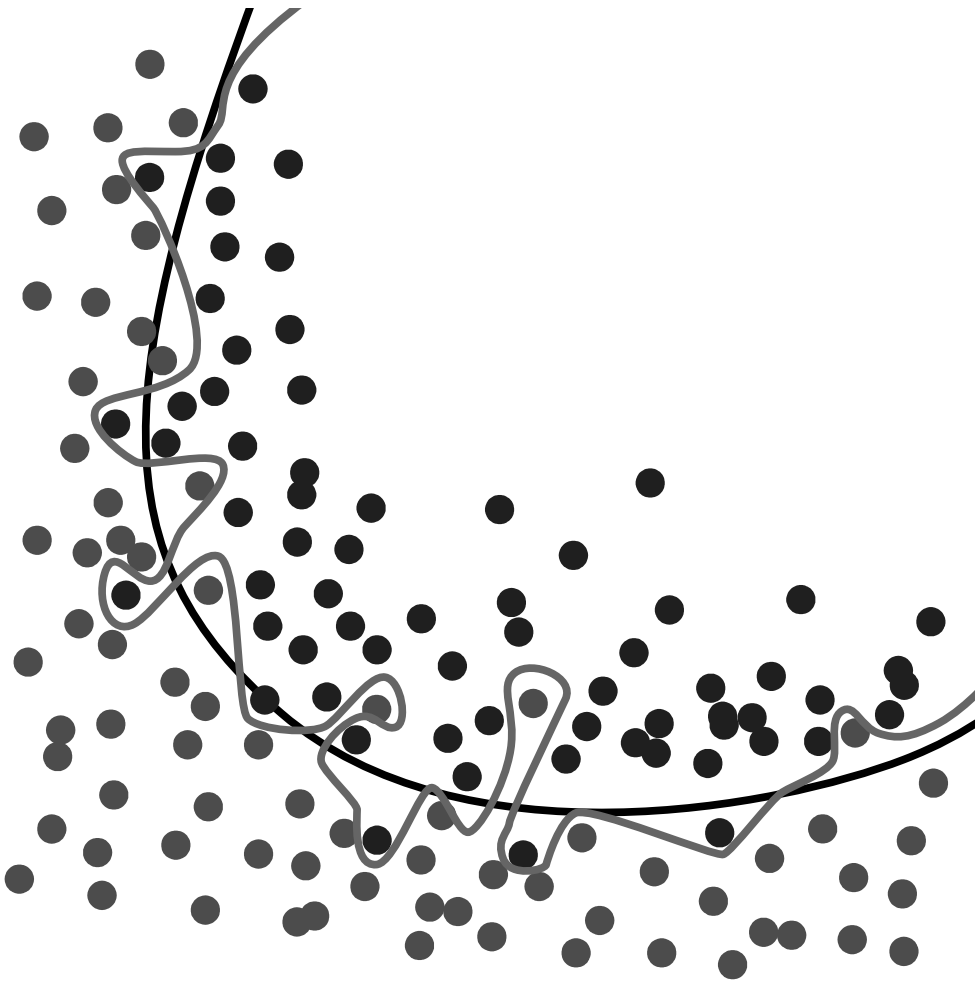
План занятия

Теоретическая часть

Переобучение и способы борьбы с ним

Переобучение

Нежелательное явление, возникающее при решении задач обучения по прецедентам, когда вероятность ошибки обученного алгоритма на объектах тестовой выборки оказывается существенно выше, чем средняя ошибка на обучающей выборке. Переобучение возникает при использовании избыточно сложных моделей.



Способы борьбы с переобучением

1. Обнаружить переобучение:
   - отложенная выборка
   - кросс-валидация
   - меры сложности модели
2. Взять больше данных
3. Взять меньше признаков
4. Выбрать более простую модель
5. Регуляризация

Регуляризация

Метод регуляризации заключается в "штрафовании" модели за слишком большие веса путем добавления нового члена к ошибке:

$$Q(w, X) + \lambda||w||^2 \to \min_w.$$

добавленный член $\lambda||w||^2$ - **квадратичный регуляризатор**, который представляет собой L_2-норму вектора весов, то есть сумму квадратов весов $\sum_{j=1}^{d} w_j^2$, коэффициент λ при нем - коэффициент регуляризации.

Чем больше его λ, тем меньшая сложность модели будет получаться в процессе такого обучения. - Если увеличивать его, в какой-то момент оптимальным для модели окажется зануление всех весов - А при слишком низких его значениях появляется вероятность чрезмерного усложнения модели и переобучения - Подбираем по кросс-валидации

По сути, смысл регуляризации заключается в минимизации функционала ошибки с ограничением весов.

$$\begin{cases} Q(w, X) \to min \\ ||w||^2 \leq C \end{cases}$$

L1-регуляризация (lasso, регуляризация через манхэттенское расстояние)

$$L_1 = \sum_{i=1} |(y_i - \widehat{y_i})| + \lambda \sum_{j=1} |w_j|$$

L2-регуляризация (ridge, регуляризация Тихонова)

$$L_2 = \sum_{i=1} (y_i - \widehat{y_i})^2 + \lambda \sum_{j=1} w_j^2$$

Практическая часть

Подключение библиотек и скриптов

Ввод [1]:
```
import pandas as pd
import numpy as np
import pickle
from pathlib import Path

from sklearn.model_selection import train_test_split, ShuffleSplit, cross_val_score, learning_curve
from sklearn.metrics import classification_report, f1_score, precision_score, recall_score
from sklearn.model_selection import StratifiedKFold, GridSearchCV, RandomizedSearchCV

from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.svm import SVC
import xgboost as xgb
import catboost as catb

import seaborn as sns
from matplotlib import pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

Ввод [2]:
```
def get_classification_report(y_train_true, y_train_pred, y_test_true, y_test_pred):
    print('TRAIN\n\n' + classification_report(y_train_true, y_train_pred))
    print('TEST\n\n' + classification_report(y_test_true, y_test_pred))
    print('CONFUSION MATRIX\n')
    print(pd.crosstab(y_test_true, y_test_pred))
```

Ввод [3]:
```
def evaluate_preds(model, X_train, X_test, y_train, y_test):
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    get_classification_report(y_train, y_train_pred, y_test, y_test_pred)
```

Ввод [4]:
```
def show_proba_calibration_plots(y_predicted_probs, y_true_labels):
    preds_with_true_labels = np.array(list(zip(y_predicted_probs, y_true_labels)))

    thresholds = []
    precisions = []
    recalls = []
    f1_scores = []

    for threshold in np.linspace(0.1, 0.9, 9):
        thresholds.append(threshold)
        precisions.append(precision_score(y_true_labels, list(map(int, y_predicted_probs > threshold))))
        recalls.append(recall_score(y_true_labels, list(map(int, y_predicted_probs > threshold))))
        f1_scores.append(f1_score(y_true_labels, list(map(int, y_predicted_probs > threshold))))

    scores_table = pd.DataFrame({'f1':f1_scores,
                                 'precision':precisions,
                                 'recall':recalls,
                                 'probability':thresholds}).sort_values('f1', ascending=False).round(3)

    figure = plt.figure(figsize = (15, 5))

    plt1 = figure.add_subplot(121)
    plt1.plot(thresholds, precisions, label='Precision', linewidth=4)
    plt1.plot(thresholds, recalls, label='Recall', linewidth=4)
    plt1.plot(thresholds, f1_scores, label='F1', linewidth=4)
    plt1.set_ylabel('Scores')
    plt1.set_xlabel('Probability threshold')
    plt1.set_title('Probabilities threshold calibration')
    plt1.legend(bbox_to_anchor=(0.25, 0.25))
    plt1.table(cellText = scores_table.values,
               colLabels = scores_table.columns,
               colLoc = 'center', cellLoc = 'center', loc = 'bottom', bbox = [0, -1.3, 1, 1])

    plt2 = figure.add_subplot(122)
    plt2.hist(preds_with_true_labels[preds_with_true_labels[:, 1] == 0][:, 0],
              label='Another class', color='royalblue', alpha=1)
    plt2.hist(preds_with_true_labels[preds_with_true_labels[:, 1] == 1][:, 0],
              label='Main class', color='darkcyan', alpha=0.8)
    plt2.set_ylabel('Number of examples')
    plt2.set_xlabel('Probabilities')
    plt2.set_title('Probability histogram')
    plt2.legend(bbox_to_anchor=(1, 1))

    plt.show()
```

Ввод [5]:
```
def show_learning_curve_plot(estimator, X, y, cv=3, n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 5)):

    train_sizes, train_scores, test_scores = learning_curve(estimator, X, y,
                                                            cv=cv,
                                                            scoring='f1',
                                                            train_sizes=train_sizes,
                                                            n_jobs=n_jobs)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.figure(figsize=(15,8))
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.title(f"Learning curves ({type(estimator).__name__})")
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    plt.legend(loc="best")
    plt.grid()
    plt.show()
```

Ввод [6]:
```
def show_feature_importances(feature_names, feature_importances, get_top=None):
    feature_importances = pd.DataFrame({'feature': feature_names, 'importance': feature_importances})
    feature_importances = feature_importances.sort_values('importance', ascending=False)

    plt.figure(figsize = (20, len(feature_importances) * 0.355))

    sns.barplot(feature_importances['importance'], feature_importances['feature'])

    plt.xlabel('Importance')
    plt.title('Importance of features')
    plt.show()

    if get_top is not None:
        return feature_importances['feature'][:get_top].tolist()
```

Ввод [13]:
```
def balance_df_by_target(df, target_name, method='over'):

    assert method in ['over', 'under', 'tomek', 'smote'], 'Неверный метод сэмплирования'

    target_counts = df[target_name].value_counts()

    major_class_name = target_counts.argmax()
    minor_class_name = target_counts.argmin()

    disbalance_coeff = int(target_counts[major_class_name] / target_counts[minor_class_name]) - 1
    if method == 'over':
        for i in range(disbalance_coeff):
            sample = df[df[target_name] == minor_class_name].sample(target_counts[minor_class_name])
            df = df.append(sample, ignore_index=True)

    elif method == 'under':
        df_ = df.copy()
        df = df_[df_[target_name] == minor_class_name]
        tmp = df_[df_[target_name] == major_class_name]
        df = df.append(tmp.iloc[
            np.random.randint(0, tmp.shape[0], target_counts[minor_class_name])
        ], ignore_index=True)

    elif method == 'tomek':
        from imblearn.under_sampling import TomekLinks
        tl = TomekLinks()
        X_tomek, y_tomek = tl.fit_sample(df.drop(columns=target_name), df[target_name])
        df = pd.concat([X_tomek, y_tomek], axis=1)

    elif method == 'smote':
        from imblearn.over_sampling import SMOTE
        smote = SMOTE()
        X_smote, y_smote = smote.fit_resample(df.drop(columns=target_name), df[target_name])
        df = pd.concat([X_smote, y_smote], axis=1)

    return df.sample(frac=1)
```

Пути к директориям и файлам

Ввод [8]:
```
DATA_ROOT = Path('./data/training_project/')
MODELS_PATH = Path('./models/')

# input
DATASET_PATH = DATA_ROOT / 'training_project_data.csv'
PREP_DATASET_PATH = DATA_ROOT / 'training_project_data_prep.csv'

# output
TRAIN_FULL_PATH = DATA_ROOT / 'training_project_train_full.csv'
TRAIN_PART_PATH = DATA_ROOT / 'training_project_train_part_b.csv'
TEST_PART_PATH = DATA_ROOT / 'training_project_test_part.csv'

SCALER_FILE_PATH = MODELS_PATH / 'scaler.pkl'
MODEL_FILE_PATH = MODELS_PATH / 'model.pkl'
```

Загрузка подготовленных датасетов

Загрузка данных

Ввод [9]:
```
df = pd.read_csv(TRAIN_FULL_PATH)
df_train = pd.read_csv(TRAIN_PART_PATH)
df_test = pd.read_csv(TEST_PART_PATH)
```

Выделение признакового описания и целевой переменной

Ввод [10]:
```
TARGET_NAME = 'NEXT_MONTH_DEFAULT'
BASE_FEATURE_NAMES = ['LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5',
                      'PAY_6', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
                      'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']

NUM_FEATURE_NAMES = ['LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6',
                     'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']

CAT_FEATURE_NAMES = ['SEX', 'EDUCATION', 'MARRIAGE', 'PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']
```

Ввод [11]:
```
X = df.drop(columns=[TARGET_NAME] + CAT_FEATURE_NAMES)
y = df[TARGET_NAME]

X_train = df_train.drop(columns=[TARGET_NAME] + CAT_FEATURE_NAMES)
y_train = df_train[TARGET_NAME]

X_test = df_test.drop(columns=[TARGET_NAME] + CAT_FEATURE_NAMES)
y_test = df_test[TARGET_NAME]
```

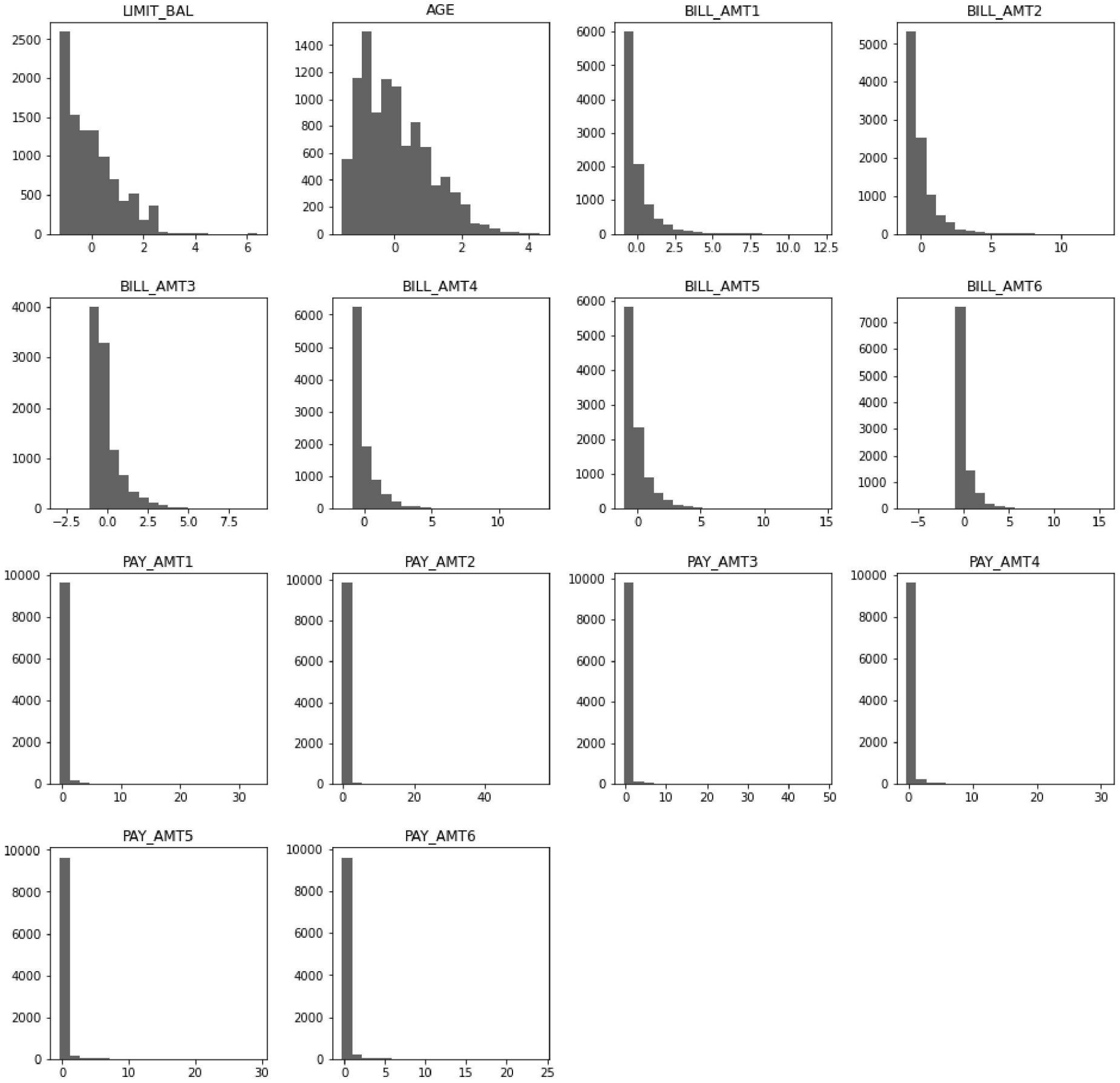
Ввод [14]:
```
df_for_balancing = pd.concat([X_train, y_train], axis=1)
df_balanced = balance_df_by_target(df_for_balancing, TARGET_NAME, method='smote')

df_balanced[TARGET_NAME].value_counts()
```

Out[14]:
```
1    5464
0    5464
Name: NEXT_MONTH_DEFAULT, dtype: int64
```

Ввод [15]:
```
X_train_balanced = df_balanced.drop(columns=TARGET_NAME)
y_train_balanced = df_balanced[TARGET_NAME]
```

Заполнение пропусков/выбросов

Ввод [16]:
```
df[NUM_FEATURE_NAMES].hist(figsize=(16, 16), bins=20, grid=False);
```



Ввод [17]:
```
df_copy = df.copy()
df_test_copy = df_test.copy()
```

Ввод [18]:
```
df_copy.isna().sum().sum()
```

Out[18]:
```
0
```

BILL_AMT_1

Ввод [19]:
```
def preprocess_outlier(df, col, threshold):
    # можно по threshold отсекать, а можно и по квантилям
    df.loc[df[col] > threshold, col] = np.nan
    return df

feature_name = 'BILL_AMT1'
df_copy = preprocess_outlier(df_copy, feature_name, threshold=5)
df_test_copy = preprocess_outlier(df_test_copy, feature_name, threshold=5)

df_copy[feature_name].isna().sum()
```

Out[19]:
```
56
```

Ввод [20]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.preprocessing import StandardScaler


def imputer_rfr(data, target_col):
    data = data.copy()

    features = data.columns

    data = data[features]

    train = data[~data[target_col].isna()]
    predict_data = data[data[target_col].isna()]

    X = train.drop(columns=target_col)
    y = train[target_col]

    X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                        test_size=0.2,
                                                        shuffle=True,
                                                        random_state=32)

    model = RandomForestRegressor(n_estimators=100,
                                  max_depth=10,
                                  random_state=42,
                                  verbose=1)
    model.fit(X_train, y_train)

    pred_train = model.predict(X_train)
    pred_test = model.predict(X_test)

    print(f"r2 на train: {r2_score(y_train, pred_train)}")
    print(f"r2 на test: {r2_score(y_test, pred_test)}")

    pred = model.predict(predict_data.drop(columns=target_col))

    data.loc[data[target_col].isna(), target_col] = list(pred)
    return model, data
```

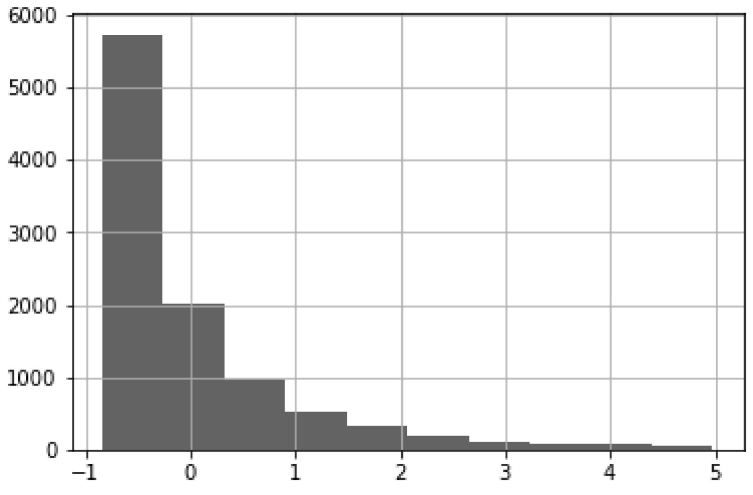
Ввод [21]:
```
%%time
bill_amt1_predictor, df_copy = imputer_rfr(df_copy, feature_name)
```

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

r2 на train: 0.9760341418393054
r2 на test: 0.9220911334360736
Wall time: 7.08 s

[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    6.7s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
```

Ввод [22]:
```
df_copy[feature_name].isna().sum()
```

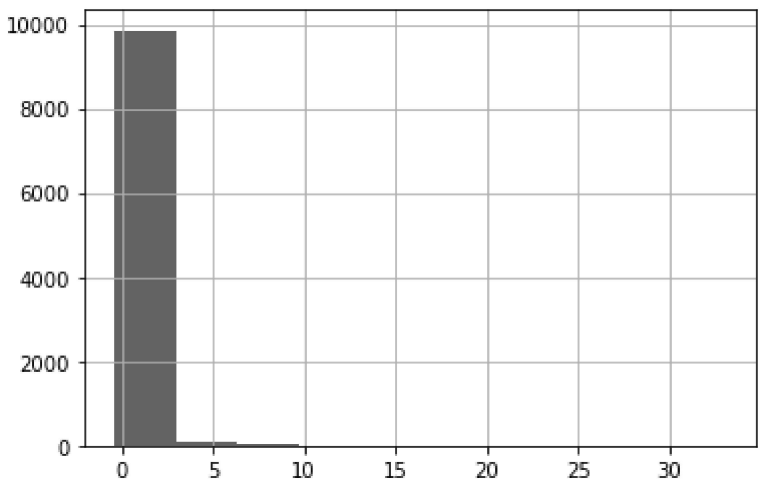
Out[22]:
```
0
```

Ввод [23]:
```
df_copy[feature_name].hist();
```



PAY_AMT1

Ввод [24]:
```
feature_name = 'PAY_AMT1'
df_copy[feature_name].hist();
```



Ввод [25]:
```
df_copy = preprocess_outlier(df_copy, feature_name, threshold=5)
df_test_copy = preprocess_outlier(df_test_copy, feature_name, threshold=5)

df_copy[feature_name].isna().sum()
```

Out[25]: 68

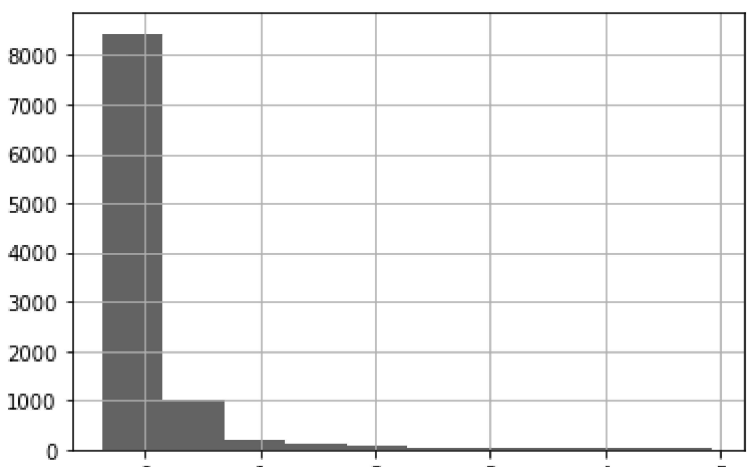
Ввод [26]:
```
pay_amt1_predictor, df_copy = imputer_rfr(df_copy, feature_name)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
r2 на train: 0.9172825911459225
r2 на test: 0.7442223568889579

[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    6.5s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.0s finished
```

Ввод [27]:
```
df_copy[feature_name].isna().sum()
```

Out[27]: 0

Ввод [28]:
```
df_copy[feature_name].hist();
```



Logistic Regression

Ввод [29]:
```
model_lr = LogisticRegression(C=0.01,
                              max_iter=7
                              )
model_lr.fit(X_train_balanced, y_train_balanced)

evaluate_preds(model_lr, X_train_balanced, X_test, y_train_balanced, y_test)
model_lr.intercept_, model_lr.coef_.mean()
```

```
TRAIN

              precision    recall  f1-score   support

           0       0.65      0.85      0.74      5464
           1       0.79      0.55      0.65      5464

    accuracy                           0.70     10928
   macro avg       0.72      0.70      0.69     10928
weighted avg       0.72      0.70      0.69     10928

TEST

              precision    recall  f1-score   support

           0       0.87      0.88      0.87      2341
           1       0.55      0.53      0.54       659

    accuracy                           0.80      3000
   macro avg       0.71      0.70      0.71      3000
weighted avg       0.80      0.80      0.80      3000

CONFUSION MATRIX

col_0                 0    1
NEXT_MONTH_DEFAULT
0                  2059  282
1                   312  347
```

Out[29]:
```
(array([0.72771297]), -0.042200650123993176)
```

Ввод [30]:
```
cv = StratifiedKFold(n_splits=3, random_state=21, shuffle=True)
```

Ввод [31]:
```
%%time
model_lr_cv = LogisticRegressionCV(max_iter=4, scoring='f1_macro',
                                   cv=cv, Cs=[0.001, 0.0001, 0.01, 0.1, 1]
                                   )
model_lr_cv.fit(X_train_balanced, y_train_balanced)

evaluate_preds(model_lr_cv, X_train_balanced, X_test, y_train_balanced, y_test)
```

```
TRAIN

              precision    recall  f1-score   support

           0       0.67      0.83      0.74      5464
           1       0.78      0.60      0.68      5464

    accuracy                           0.71     10928
   macro avg       0.73      0.71      0.71     10928
weighted avg       0.73      0.71      0.71     10928

TEST

              precision    recall  f1-score   support

           0       0.87      0.86      0.86      2341
           1       0.52      0.56      0.54       659

    accuracy                           0.79      3000
   macro avg       0.70      0.71      0.70      3000
weighted avg       0.79      0.79      0.79      3000

CONFUSION MATRIX

col_0                 0    1
NEXT_MONTH_DEFAULT
0                  2003  338
1                   293  366
Wall time: 210 ms
```

Ввод [32]:
```
model_lr_cv.Cs_
```

Out[32]:
```
array([1.e-03, 1.e-04, 1.e-02, 1.e-01, 1.e+00])
```

Ввод [33]:
```
model_lr_cv.C_
```

Out[33]:
```
array([1.])
```

Support Vector Machine

Ввод [34]:
```
%%time
model_svc = SVC(C=0.05,
                kernel='rbf'
                )
model_svc.fit(X_train_balanced, y_train_balanced)

evaluate_preds(model_svc, X_train_balanced, X_test, y_train_balanced, y_test)
```

```
TRAIN

              precision    recall  f1-score   support

           0       0.66      0.87      0.75      5464
           1       0.81      0.55      0.66      5464

    accuracy                           0.71     10928
   macro avg       0.73      0.71      0.70     10928
weighted avg       0.73      0.71      0.70     10928

TEST

              precision    recall  f1-score   support

           0       0.86      0.88      0.87      2341
           1       0.54      0.50      0.52       659

    accuracy                           0.80      3000
   macro avg       0.70      0.69      0.70      3000
weighted avg       0.79      0.80      0.79      3000

CONFUSION MATRIX

col_0                  0    1
NEXT_MONTH_DEFAULT
0                   2066  275
1                    332  327
Wall time: 26.4 s
```

Ввод [35]:
```
%%time
model_svc = SVC(C=0.1,
                kernel='poly',
                degree=3
                )
model_svc.fit(X_train_balanced, y_train_balanced)

evaluate_preds(model_svc, X_train_balanced, X_test, y_train_balanced, y_test)
```

```
TRAIN

              precision    recall  f1-score   support

           0       0.70      0.84      0.76      5464
           1       0.80      0.64      0.71      5464

    accuracy                           0.74     10928
   macro avg       0.75      0.74      0.74     10928
weighted avg       0.75      0.74      0.74     10928

TEST

              precision    recall  f1-score   support

           0       0.87      0.85      0.86      2341
           1       0.50      0.54      0.52       659

    accuracy                           0.78      3000
   macro avg       0.69      0.70      0.69      3000
weighted avg       0.79      0.78      0.79      3000

CONFUSION MATRIX

col_0                  0    1
NEXT_MONTH_DEFAULT
0                   1990  351
1                    301  358
Wall time: 14.1 s
```

XGBoost

Ввод [39]:
```
model_xgb = xgb.XGBClassifier(random_state=21,
                              max_depth=1,
                              )
model_xgb.fit(X_train_balanced, y_train_balanced)

evaluate_preds(model_xgb, X_train_balanced, X_test, y_train_balanced, y_test)
```

```
[20:34:27] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoos
t 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'.
Explicitly set eval_metric if you'd like to restore the old behavior.
TRAIN

              precision    recall  f1-score   support

           0       0.71      0.82      0.76      5464
           1       0.78      0.66      0.72      5464

    accuracy                           0.74     10928
   macro avg       0.75      0.74      0.74     10928
weighted avg       0.75      0.74      0.74     10928

TEST

              precision    recall  f1-score   support

           0       0.88      0.84      0.86      2341
           1       0.51      0.58      0.55       659

    accuracy                           0.79      3000
   macro avg       0.70      0.71      0.70      3000
weighted avg       0.80      0.79      0.79      3000

CONFUSION MATRIX

col_0                  0    1
NEXT_MONTH_DEFAULT
0                   1976  365
1                    275  384
```

Ввод [40]:
```
params = {
    'reg_lambda': [0.2, 0.5, 0.9, 1.3, 1.9, 3, 5, 10]
}
```

Ввод [41]:
```
cv = StratifiedKFold(n_splits=3, random_state=21, shuffle=True)
```

Ввод [42]:
```
%%time
grid_search = GridSearchCV(param_grid=params, estimator=model_xgb, cv=cv, verbose=0, scoring='f1')
grid_search.fit(X_train_balanced, y_train_balanced)
display(grid_search.best_params_, grid_search.best_score_)
```

```
[20:34:40] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBo
ost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
[20:34:40] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBo
ost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
[20:34:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBo
ost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
[20:34:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBo
ost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.

{'reg_lambda': 0.9}

0.713099507797911

Wall time: 5.32 s
```

Ввод [43]:
```
model_xgb = xgb.XGBClassifier(random_state=21,
                              max_depth=1,
                              reg_lambda=0.5
                             )
model_xgb.fit(X_train_balanced, y_train_balanced)

evaluate_preds(model_xgb, X_train_balanced, X_test, y_train_balanced, y_test)
```

```
[20:34:56] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoos
t 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'.
Explicitly set eval_metric if you'd like to restore the old behavior.
TRAIN

              precision    recall  f1-score   support

           0       0.71      0.82      0.76      5464
           1       0.78      0.66      0.72      5464

    accuracy                           0.74     10928
   macro avg       0.75      0.74      0.74     10928
weighted avg       0.75      0.74      0.74     10928

TEST

              precision    recall  f1-score   support

           0       0.88      0.84      0.86      2341
           1       0.51      0.58      0.55       659

    accuracy                           0.79      3000
   macro avg       0.70      0.71      0.70      3000
weighted avg       0.80      0.79      0.79      3000

CONFUSION MATRIX

col_0                  0    1
NEXT_MONTH_DEFAULT
0                   1978  363
1                    276  383
```

Подбор оптимального размера выборки, кривая обучения

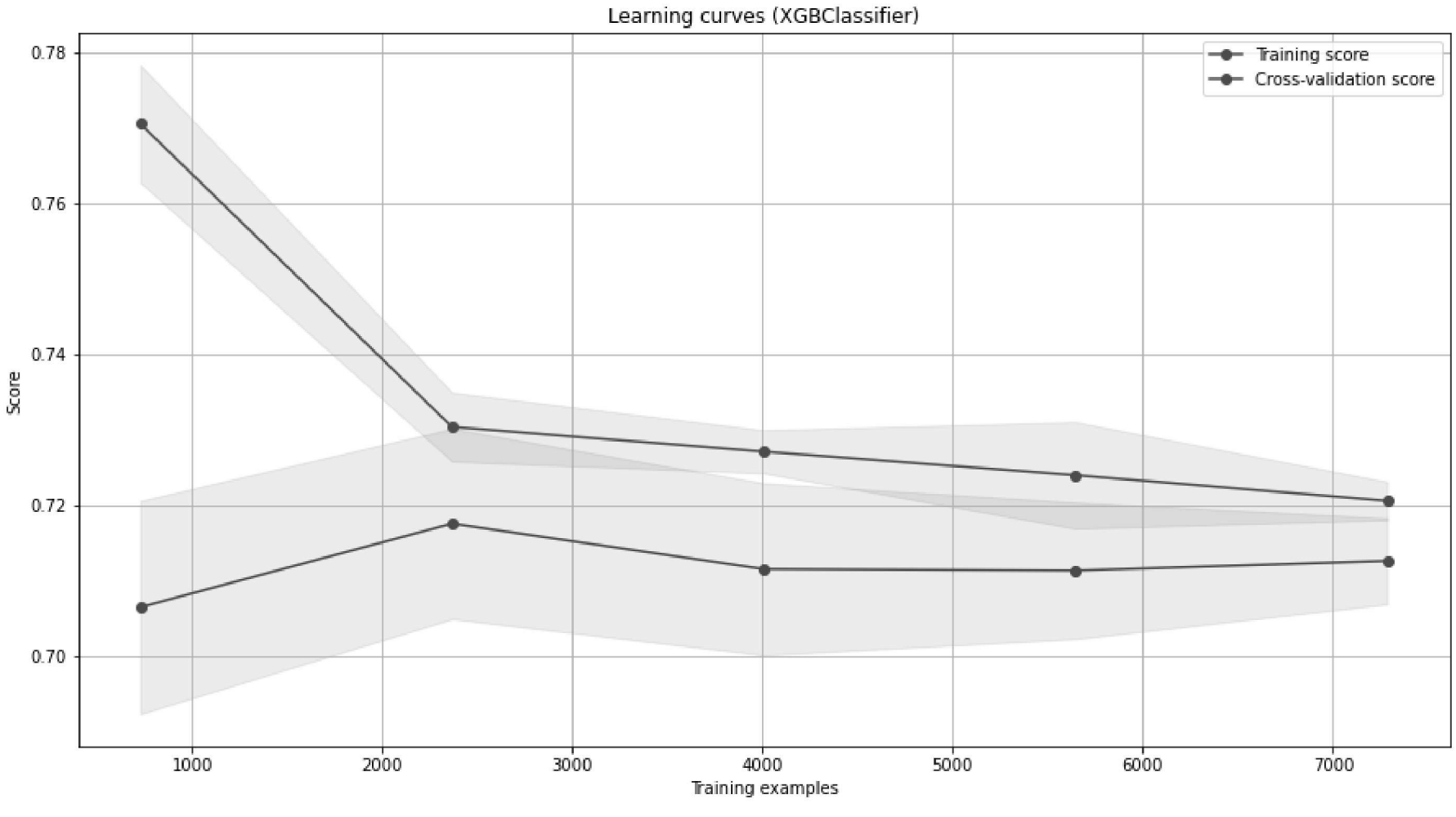
Ввод [44]:
```
import inspect
print(inspect.getsource(show_learning_curve_plot))
```

```
def show_learning_curve_plot(estimator, X, y, cv=3, n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 5)):

    train_sizes, train_scores, test_scores = learning_curve(estimator, X, y,
                                                            cv=cv,
                                                            scoring='f1',
                                                            train_sizes=train_sizes,
                                                            n_jobs=n_jobs)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.figure(figsize=(15,8))
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.title(f"Learning curves ({type(estimator).__name__})")
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    plt.legend(loc="best")
    plt.grid()
    plt.show()
```

Ввод [45]:
```
np.linspace(.1, 1.0, 5)
```

Out[45]:
```
array([0.1  , 0.325, 0.55 , 0.775, 1.   ])
```

Ввод [46]:
```
show_learning_curve_plot(model_xgb, X_train_balanced, y_train_balanced)
```



Выбор порога вероятности для определения класса

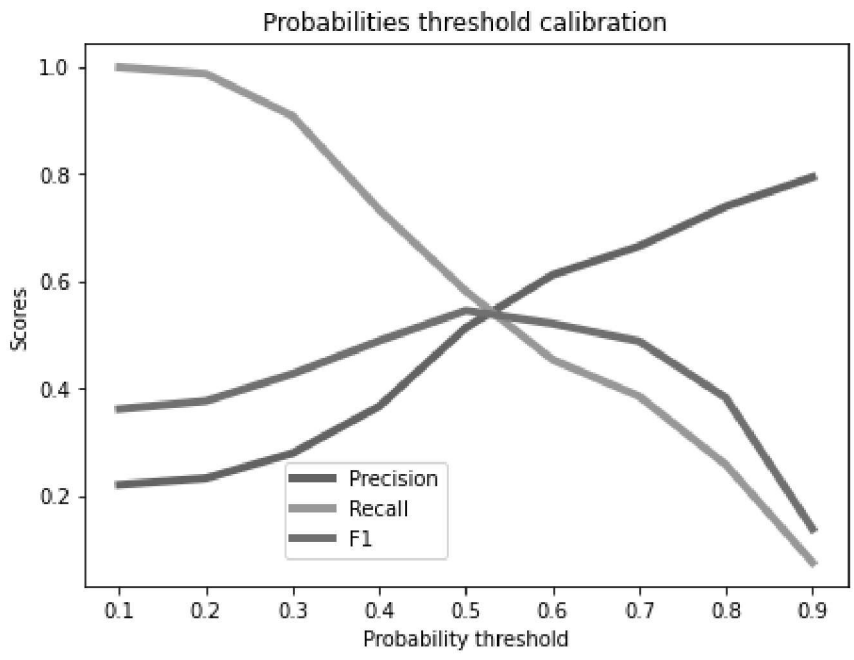
Ввод [47]:
```
y_test_pred_probs = model_xgb.predict_proba(X_test)
y_test_pred_probs
```

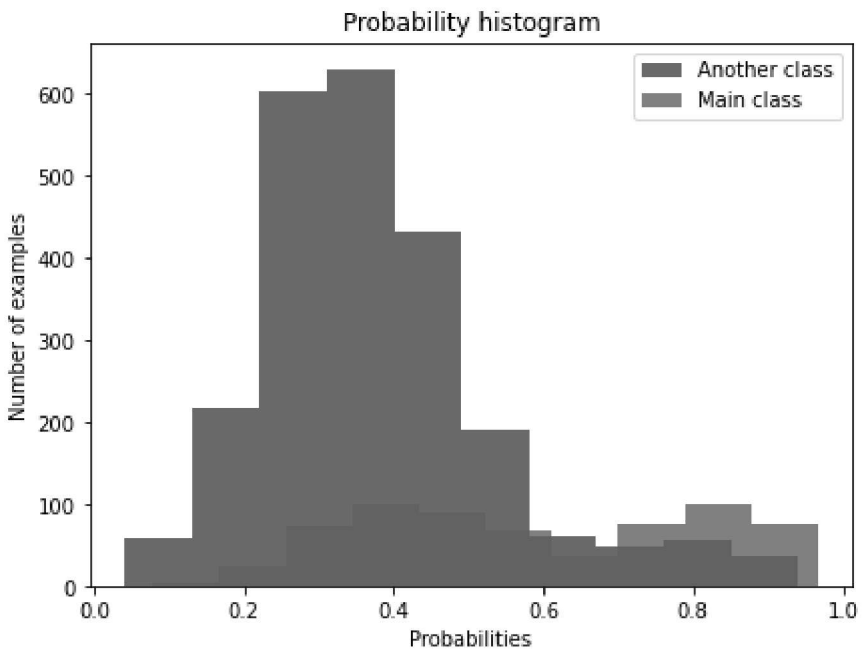
Out[47]:
```
array([[0.70677644, 0.29322356],
       [0.6515911 , 0.34840888],
       [0.5780883 , 0.42191172],
       ...,
       [0.7057123 , 0.29428765],
       [0.21124679, 0.7887532 ],
       [0.72217846, 0.2778215 ]], dtype=float32)
```

Ввод [48]:
```
print(inspect.getsource(show_proba_calibration_plots))
```

```
def show_proba_calibration_plots(y_predicted_probs, y_true_labels):
    preds_with_true_labels = np.array(list(zip(y_predicted_probs, y_true_labels)))

    thresholds = []
    precisions = []
    recalls = []
    f1_scores = []

    for threshold in np.linspace(0.1, 0.9, 9):
        thresholds.append(threshold)
        precisions.append(precision_score(y_true_labels, list(map(int, y_predicted_probs > threshold))))
        recalls.append(recall_score(y_true_labels, list(map(int, y_predicted_probs > threshold))))
        f1_scores.append(f1_score(y_true_labels, list(map(int, y_predicted_probs > threshold))))

    scores_table = pd.DataFrame({'f1':f1_scores,
                                 'precision':precisions,
                                 'recall':recalls,
                                 'probability':thresholds}).sort_values('f1', ascending=False).round(3)

    figure = plt.figure(figsize = (15, 5))

    plt1 = figure.add_subplot(121)
    plt1.plot(thresholds, precisions, label='Precision', linewidth=4)
    plt1.plot(thresholds, recalls, label='Recall', linewidth=4)
    plt1.plot(thresholds, f1_scores, label='F1', linewidth=4)
    plt1.set_ylabel('Scores')
    plt1.set_xlabel('Probability threshold')
    plt1.set_title('Probabilities threshold calibration')
    plt1.legend(bbox_to_anchor=(0.25, 0.25))
    plt1.table(cellText = scores_table.values,
               colLabels = scores_table.columns,
               colLoc = 'center', cellLoc = 'center', loc = 'bottom', bbox = [0, -1.3, 1, 1])

    plt2 = figure.add_subplot(122)
    plt2.hist(preds_with_true_labels[preds_with_true_labels[:, 1] == 0][:, 0],
              label='Another class', color='royalblue', alpha=1)
    plt2.hist(preds_with_true_labels[preds_with_true_labels[:, 1] == 1][:, 0],
              label='Main class', color='darkcyan', alpha=0.8)
    plt2.set_ylabel('Number of examples')
    plt2.set_xlabel('Probabilities')
    plt2.set_title('Probability histogram')
    plt2.legend(bbox_to_anchor=(1, 1))

    plt.show()
```

Ввод [49]:
```
show_proba_calibration_plots(y_test_pred_probs[:, 1], y_test)
```





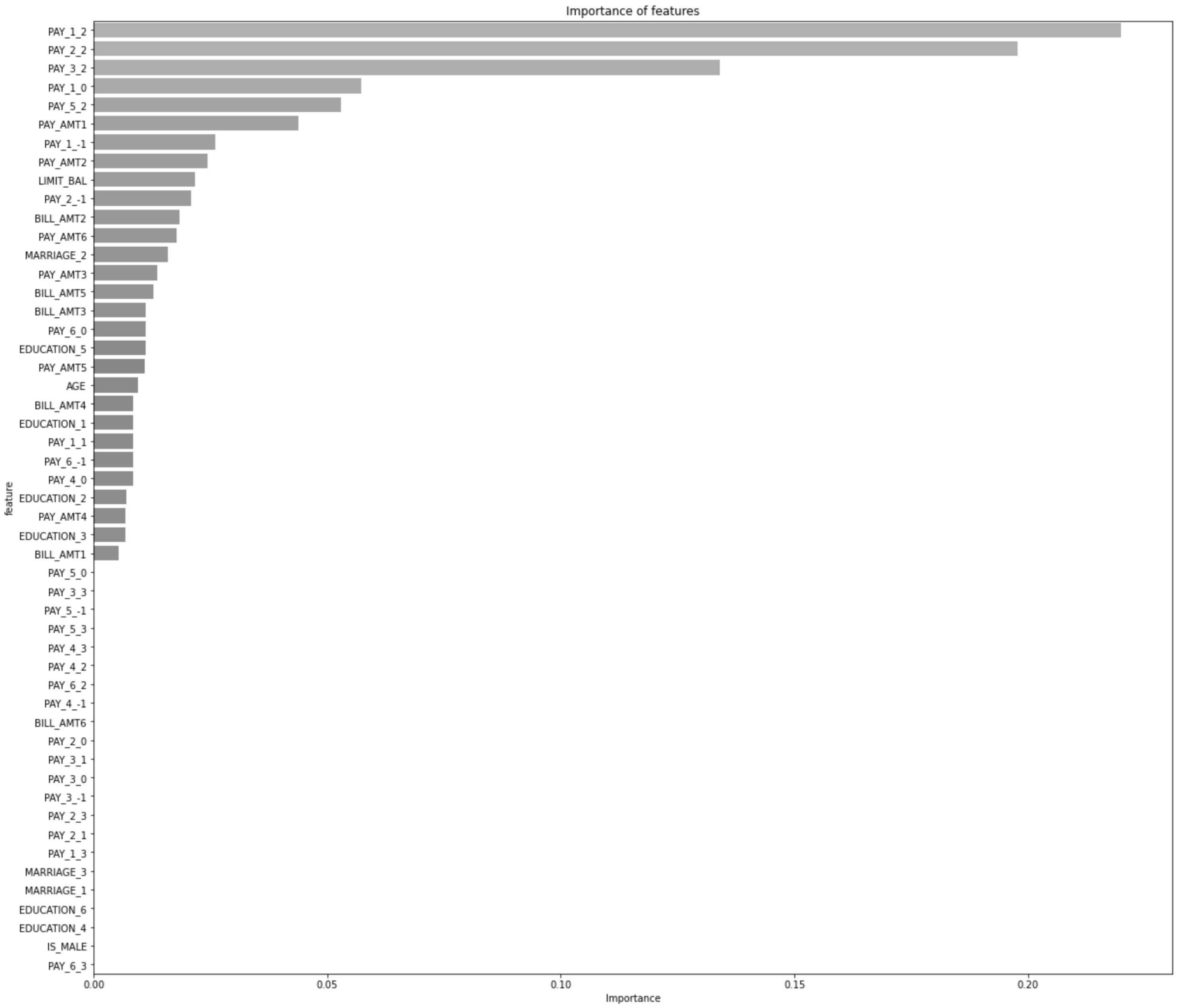
f1	precision	recall	probability
0.545	0.513	0.581	0.5
0.521	0.611	0.454	0.6
0.489	0.367	0.733	0.4
0.488	0.665	0.385	0.7
0.427	0.279	0.907	0.3
0.382	0.739	0.258	0.8
0.377	0.233	0.986	0.2
0.362	0.221	0.998	0.1
0.139	0.794	0.076	0.9

Важность признаков

Ввод [50]:
```
print(inspect.getsource(show_feature_importances))
```

```
def show_feature_importances(feature_names, feature_importances, get_top=None):
    feature_importances = pd.DataFrame({'feature': feature_names, 'importance': feature_importances})
    feature_importances = feature_importances.sort_values('importance', ascending=False)

    plt.figure(figsize = (20, len(feature_importances) * 0.355))

    sns.barplot(feature_importances['importance'], feature_importances['feature'])

    plt.xlabel('Importance')
    plt.title('Importance of features')
    plt.show()

    if get_top is not None:
        return feature_importances['feature'][:get_top].tolist()
```

Ввод [51]:
```
important_features_top = show_feature_importances(X_train_balanced.columns,
                                                  model_xgb.feature_importances_, get_top=19)
```



Ввод [52]:
```
important_features_top
```

Out[52]:
```
['PAY_1_2',
 'PAY_2_2',
 'PAY_3_2',
 'PAY_1_0',
 'PAY_5_2',
 'PAY_AMT1',
 'PAY_1_-1',
 'PAY_AMT2',
 'LIMIT_BAL',
 'PAY_2_-1',
 'BILL_AMT2',
 'PAY_AMT6',
 'MARRIAGE_2',
 'PAY_AMT3',
 'BILL_AMT5',
 'BILL_AMT3',
 'PAY_6_0',
 'EDUCATION_5',
 'PAY_AMT5']
```

Финальная модель

Ввод [53]:
```
final_xgb = xgb.XGBClassifier(random_state=21,
                              max_depth=1,
                              reg_lambda=0.5
                              )
final_xgb.fit(X_train_balanced[important_features_top], y_train_balanced)

evaluate_preds(final_xgb,
               X_train_balanced[important_features_top],
               X_test[important_features_top],
               y_train_balanced,
               y_test)
```

```
[20:35:32] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoos
t 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'.
Explicitly set eval_metric if you'd like to restore the old behavior.
TRAIN

              precision    recall  f1-score   support

           0       0.71      0.80      0.75      5464
           1       0.77      0.67      0.72      5464

    accuracy                           0.74     10928
   macro avg       0.74      0.74      0.74     10928
weighted avg       0.74      0.74      0.74     10928

TEST

              precision    recall  f1-score   support

           0       0.88      0.83      0.85      2341
           1       0.49      0.59      0.54       659

    accuracy                           0.78      3000
   macro avg       0.69      0.71      0.70      3000
weighted avg       0.79      0.78      0.78      3000

CONFUSION MATRIX

col_0                   0    1
NEXT_MONTH_DEFAULT
0                    1942  399
1                     270  389
```

Сохранение финальной модели

Ввод [54]:
```
with open(MODEL_FILE_PATH, 'wb') as file:
    pickle.dump(final_xgb, file)
```

Catboost

Ввод [55]:
```
X = df[BASE_FEATURE_NAMES]
y = df[TARGET_NAME]

X_train = df_train[BASE_FEATURE_NAMES]
y_train = df_train[TARGET_NAME]

X_test = df_test[BASE_FEATURE_NAMES]
y_test = df_test[TARGET_NAME]
```

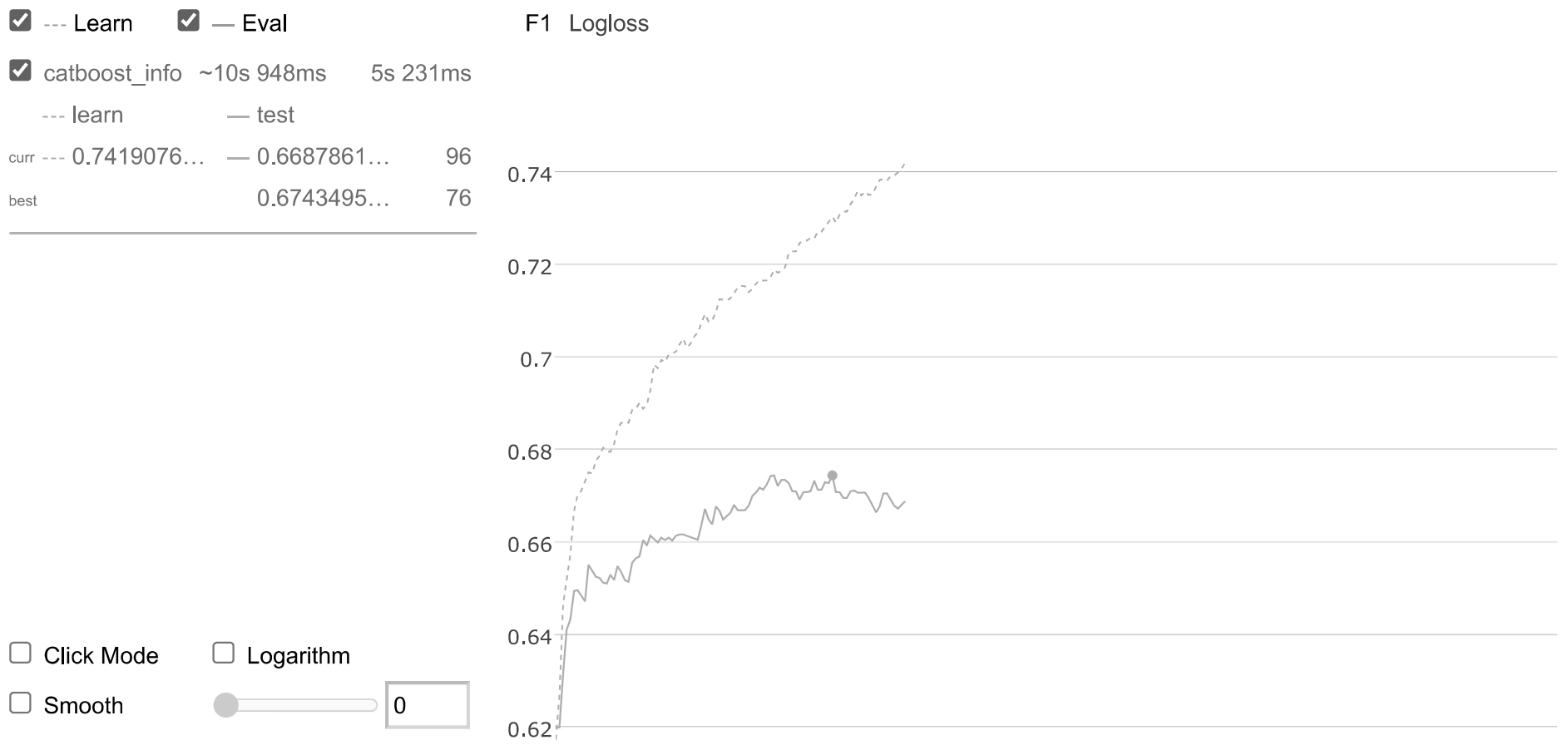
Модель, полученная на предыдущем уроке

Ввод [56]:
```
disbalance = y_train.value_counts()[0] / y_train.value_counts()[1]
disbalance
```

Out[56]:
```
3.5572916666666665
```

Ввод [57]:
```
frozen_params = {
     'class_weights':[1, disbalance],
     'silent':True,
     'random_state':21,
     'cat_features':CAT_FEATURE_NAMES,
     'eval_metric':'F1',
     'early_stopping_rounds':20
}
```

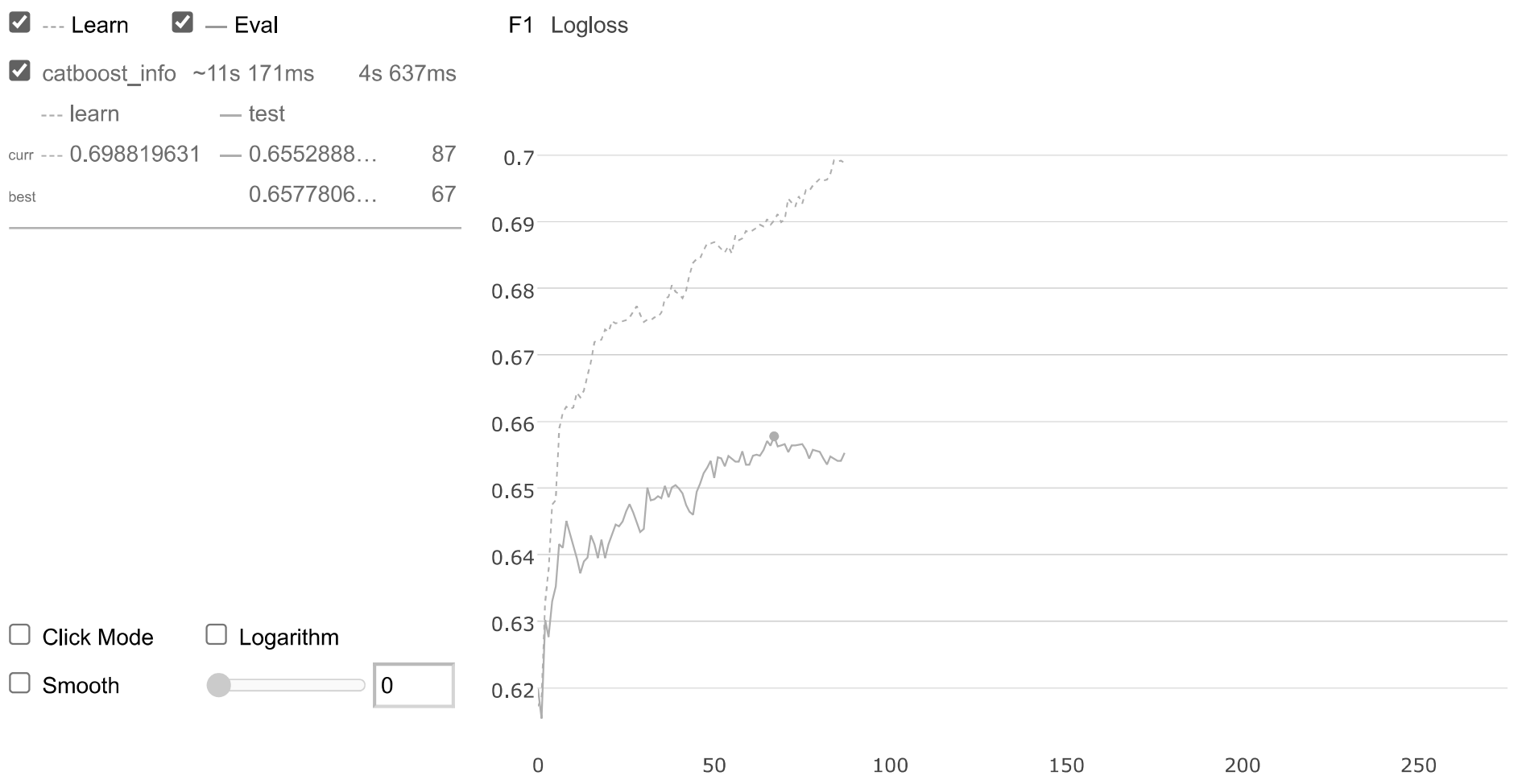
Ввод [58]:
```
%%time

model_catb = catb.CatBoostClassifier(**frozen_params, iterations=300, max_depth=7)
model_catb.fit(X_train, y_train, plot=True, eval_set=(X_test, y_test))

evaluate_preds(model_catb, X_train, X_test, y_train, y_test)
```




```
TRAIN

              precision    recall  f1-score   support

           0       0.90      0.83      0.86      5464
           1       0.52      0.67      0.58      1536

    accuracy                           0.79      7000
   macro avg       0.71      0.75      0.72      7000
weighted avg       0.81      0.79      0.80      7000

TEST

              precision    recall  f1-score   support

           0       0.88      0.83      0.85      2341
           1       0.50      0.59      0.54       659

    accuracy                           0.78      3000
   macro avg       0.69      0.71      0.70      3000
weighted avg       0.80      0.78      0.79      3000

CONFUSION MATRIX

col_0                0    1
NEXT MONTH DEFAULT
0                 1944  397
1                  267  392
Wall time: 5.58 s
```

Снижение степени переобучения

Ввод [59]:
```
model = catb.CatBoostClassifier(**frozen_params,
                                iterations=300,
                                max_depth=7,
#                                 l2_leaf_reg=.5,
                                reg_lambda=0.5
                                )

model.fit(X_train, y_train, plot=True, eval_set=(X_test, y_test))

evaluate_preds(model, X_train, X_test, y_train, y_test)
```



TRAIN

	precision	recall	f1-score	support
0	0.88	0.84	0.86	5464
1	0.52	0.61	0.56	1536
accuracy			0.79	7000
macro avg	0.70	0.73	0.71	7000
weighted avg	0.80	0.79	0.80	7000

TEST

	precision	recall	f1-score	support
0	0.87	0.85	0.86	2341
1	0.51	0.56	0.54	659
accuracy			0.79	3000
macro avg	0.69	0.71	0.70	3000
weighted avg	0.79	0.79	0.79	3000

CONFUSION MATRIX

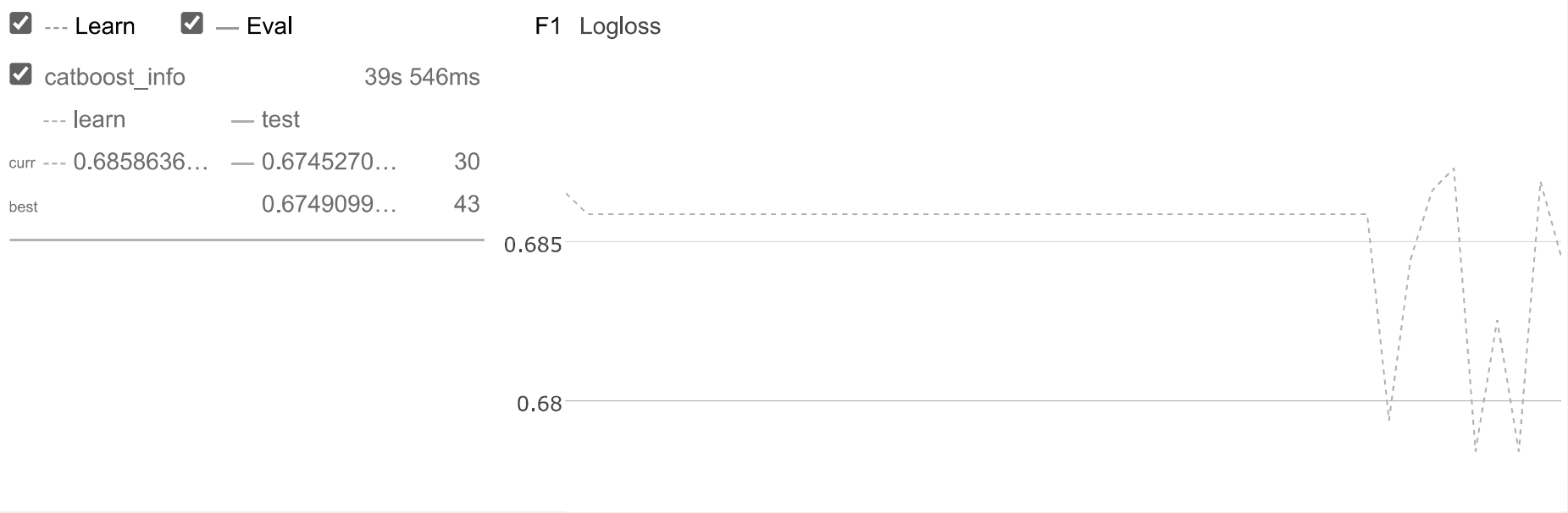
col_0	0	1
NEXT_MONTH_DEFAULT		
0	1985	356
1	287	372

Ввод [60]:
```
params = {
    'reg_lambda': np.linspace(0.1, 4, 80)
}
```

Ввод [61]:
```
model = catb.CatBoostClassifier(**frozen_params,
                                iterations=300,
                                max_depth=7)
```

Ввод [62]:
```
cv = StratifiedKFold(n_splits=3, random_state=21, shuffle=True)
```

Ввод [63]:
```
grid_search = model.randomized_search(params, X_train, y_train, n_iter=50, cv=cv, stratified=True, plot=True, refit=True)
```



Ввод [64]:
```
grid_search
```

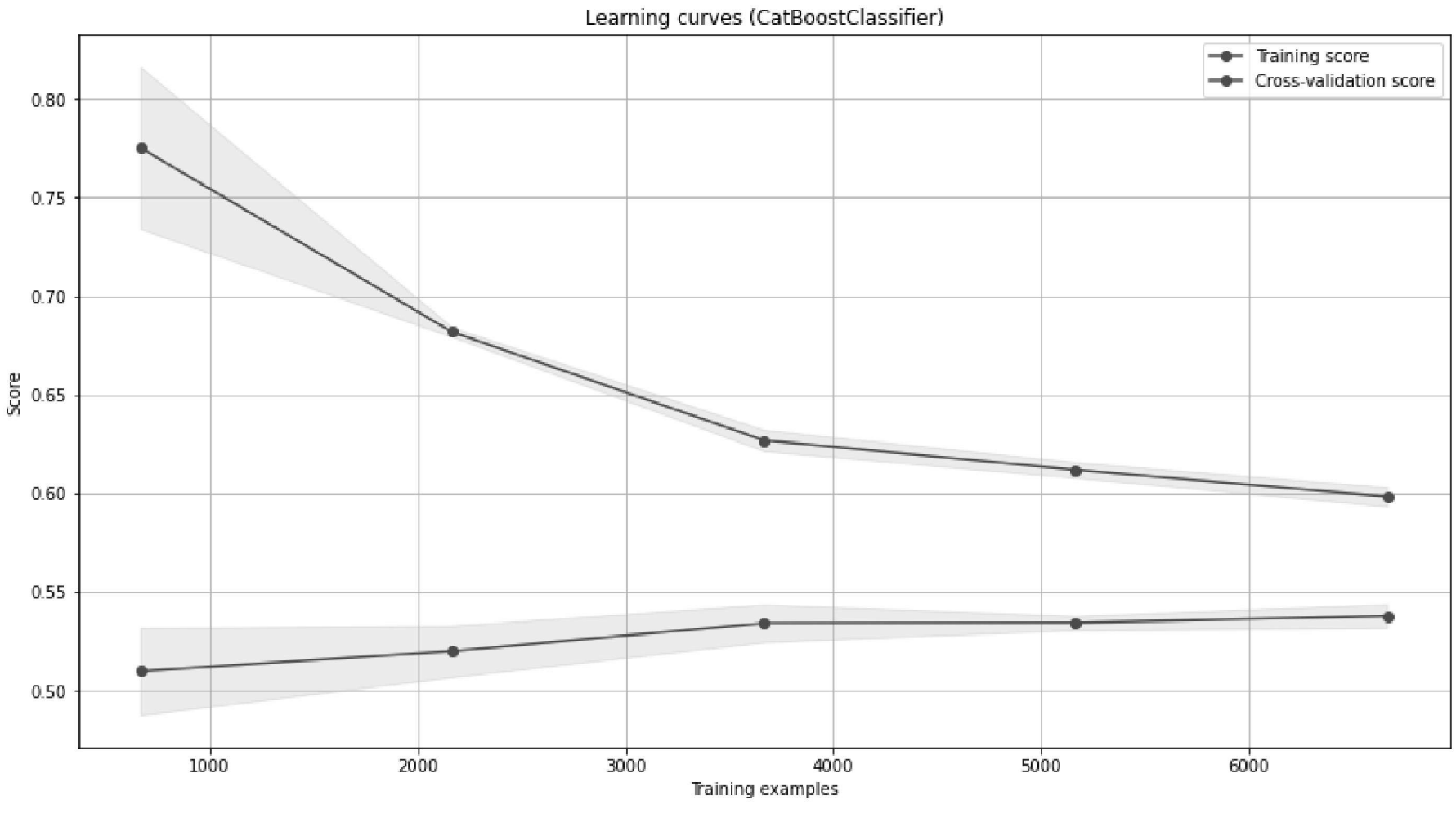
```
                  0.0021965462193440513,
                  0.0016453528550982857,
                  0.0020593190465724256,
                  0.0020822580414864475,
                  0.0024269506407092536,
                  0.0026786796827382095,
                  0.0028451225436176984,
                  0.00294500764213077,
                  0.003002694999064046,
                  0.002954905879036406,
                  0.0029465836387492245,
                  0.0033299368937180643,
                  0.0035494735912875996,
                  0.003400748193450346,
                  0.003598375467689835,
                  0.0035693383691929818,
                  0.0035783747051118236,
                  0.003607416565131993,
                  0.003549356890977293,
                  0.0034517714109757191]})}
```

Ввод [65]:
```
%%time

evaluate_preds(model, X_train, X_test, y_train, y_test)
```

```
TRAIN

              precision    recall  f1-score   support

           0       0.90      0.82      0.86      5464
           1       0.52      0.68      0.59      1536

    accuracy                           0.79      7000
   macro avg       0.71      0.75      0.73      7000
weighted avg       0.82      0.79      0.80      7000

TEST

              precision    recall  f1-score   support

           0       0.88      0.82      0.85      2341
           1       0.49      0.59      0.54       659

    accuracy                           0.77      3000
   macro avg       0.68      0.71      0.69      3000
weighted avg       0.79      0.77      0.78      3000

CONFUSION MATRIX

col_0                   0    1
NEXT_MONTH_DEFAULT
0                    1927  414
1                     267  392
Wall time: 96.2 ms
```

Подбор оптимального размера выборки, кривая обучения

Ввод [66]:
```
show_learning_curve_plot(model, X, y)
```

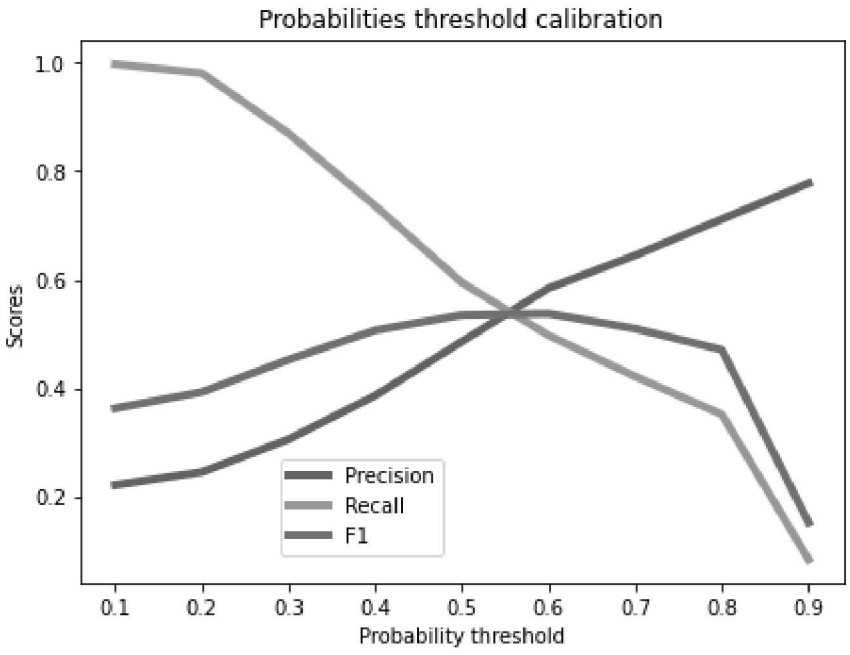


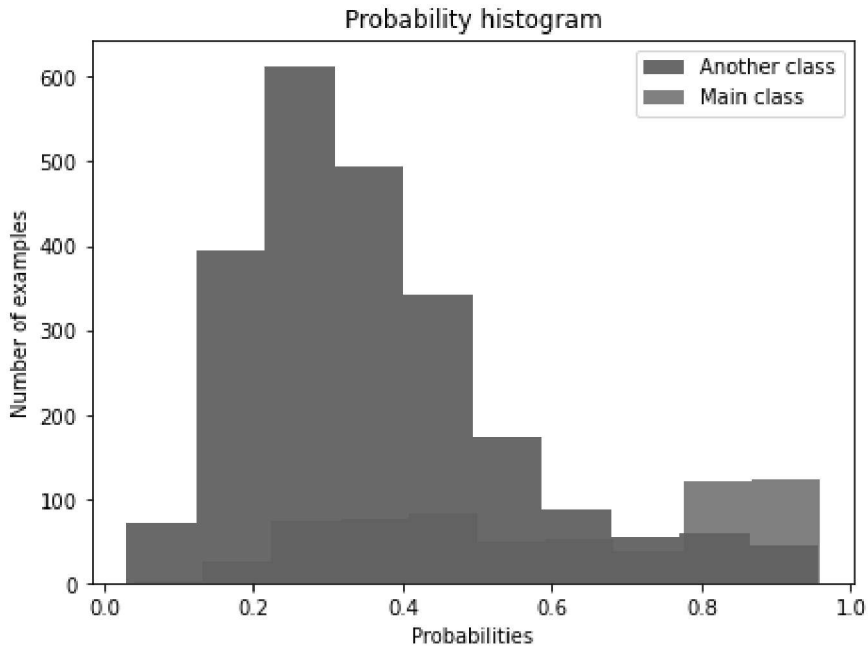
Выбор порога вероятности для определения класса

Ввод [67]:
```
np.where(y_test_pred_probs[:, 1] > 0.6, 1, 0)
```

Out[67]:
```
array([0, 0, 0, ..., 0, 1, 0])
```

Ввод [68]:
```
y_test_pred_probs = model.predict_proba(X_test)
show_proba_calibration_plots(y_test_pred_probs[:, 1], y_test)
```

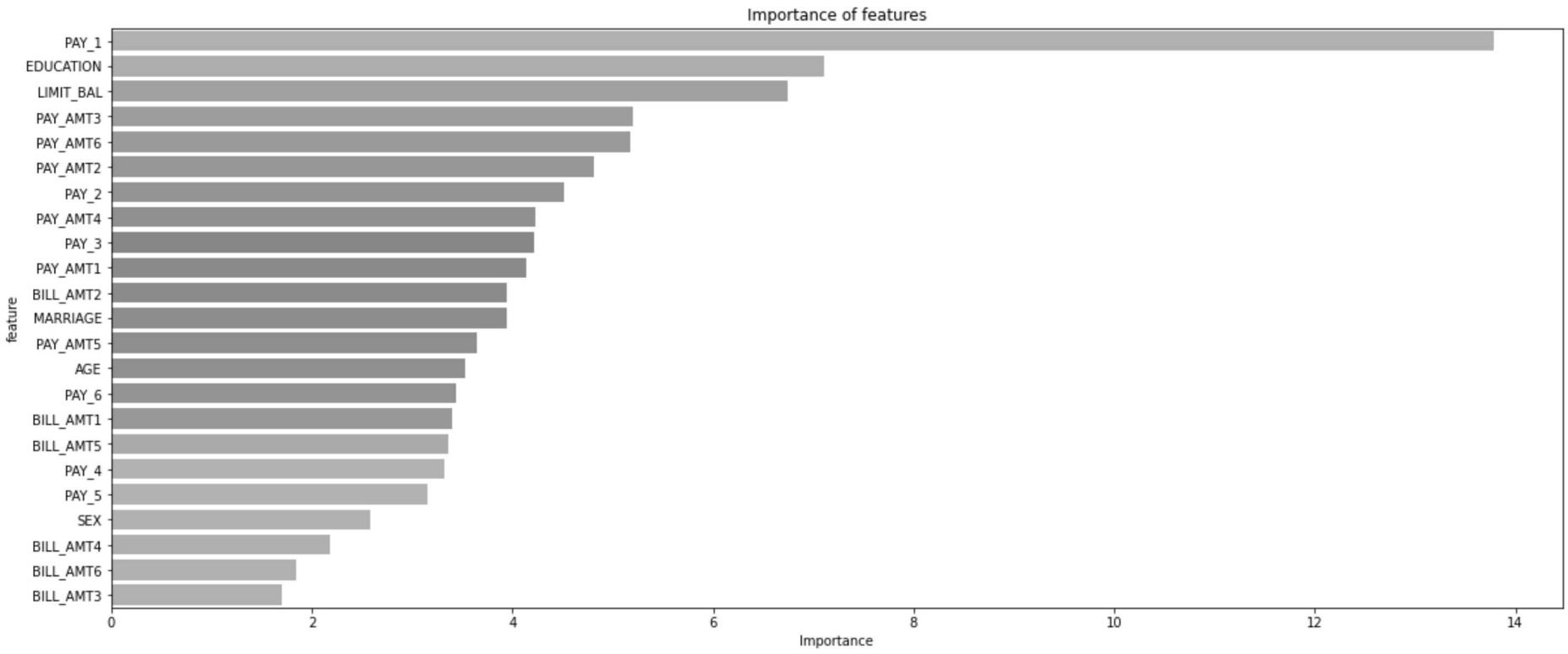




f1	precision	recall	probability
0.538	0.585	0.498	0.6
0.535	0.486	0.595	0.5
0.51	0.645	0.422	0.7
0.507	0.387	0.736	0.4
0.471	0.712	0.352	0.8
0.453	0.306	0.869	0.3
0.393	0.246	0.98	0.2
0.363	0.222	0.997	0.1
0.153	0.778	0.085	0.9

Важность признаков

Ввод [69]:
```
important_features_top = show_feature_importances(X_train.columns, model.feature_importances_, get_top=23)
```



Ввод [70]:
```
important_features_top
```

Out[70]:
```
['PAY_1',
 'EDUCATION',
 'LIMIT_BAL',
 'PAY_AMT3',
 'PAY_AMT6',
 'PAY_AMT2',
 'PAY_2',
 'PAY_AMT4',
 'PAY_3',
 'PAY_AMT1',
 'BILL_AMT2',
 'MARRIAGE',
 'PAY_AMT5',
 'AGE',
 'PAY_6',
 'BILL_AMT1',
 'BILL_AMT5',
 'PAY_4',
 'PAY_5',
 'SEX',
 'BILL_AMT4',
 'BILL_AMT6',
 'BILL_AMT3']
```

Финальная модель

Ввод [71]:
```
CAT_FEATURE_NAMES
```

Out[71]:
```
['SEX',
 'EDUCATION',
 'MARRIAGE',
 'PAY_1',
 'PAY_2',
 'PAY_3',
 'PAY_4',
 'PAY_5',
 'PAY_6']
```

Ввод [72]:
```
NEW_CAT_FEATURE_NAMES = list(set(CAT_FEATURE_NAMES).intersection(set(important_features_top)))
NEW_CAT_FEATURE_NAMES
```

Out[72]:
```
['PAY_6',
 'PAY_4',
 'PAY_2',
 'PAY_5',
 'PAY_3',
 'SEX',
 'EDUCATION',
 'MARRIAGE',
 'PAY_1']
```

Ввод [73]:
```
%%time

frozen_params = {
     'class_weights':[1, disbalance],
     'silent':True,
     'random_state':21,
     'cat_features':NEW_CAT_FEATURE_NAMES,
     'eval_metric':'F1',
     'early_stopping_rounds':20
}

final_model = catb.CatBoostClassifier(**frozen_params,
                                      iterations=300,
                                      max_depth=7,
                                      reg_lambda=0.5)

final_model.fit(X_train, y_train, eval_set=(X_test, y_test))

evaluate_preds(final_model, X_train, X_test, y_train, y_test)
```

```
TRAIN

              precision    recall  f1-score   support

           0       0.88      0.84      0.86      5464
           1       0.52      0.61      0.56      1536

    accuracy                           0.79      7000
   macro avg       0.70      0.73      0.71      7000
weighted avg       0.80      0.79      0.80      7000

TEST

              precision    recall  f1-score   support

           0       0.87      0.85      0.86      2341
           1       0.51      0.56      0.54       659

    accuracy                           0.79      3000
   macro avg       0.69      0.71      0.70      3000
weighted avg       0.79      0.79      0.79      3000

CONFUSION MATRIX

col_0                   0    1
NEXT_MONTH_DEFAULT
0                    1985  356
1                     287  372
Wall time: 4.46 s
```

Сохранение финальной модели

Ввод [74]:
```
with open(MODEL_FILE_PATH, 'wb') as file:
    pickle.dump(final_model, file)
```

Курсовой проект

https://www.kaggle.com/c/gb-credit-default/ (https://www.kaggle.com/c/gb-credit-default/)

- F1-score > 0.5 на private Leaderboard
- F1-score_1, recall_1, precision_1 > 0.5
- Решение прикреплять в ДЗ к Урок 4. Оценка и интерпретация полученной модели. Обсуждение курсового проекта.
- Указать свой ник на kaggle

Обобщение работы с новой задачей

1. Получить базовое решение * Минимально познакомиться с данными * Заполнить пропуски простым методом (нулями, медиана, среднее, мода...) * Обучить простую модель (линейная, деревья...) * Посчитать метрику качества **2. EDA** * Изучить целевую переменную * регрессия - распределение, меры центральной тенденции * классификация - баланс классов * Изучить признаки * корреляция * найти проблемные признаки * найти пропуски * найти выбросы * сгенерировать идеи по их заполнению * Изучить влияние признаков на целевую переменную * корреляция * классификация - разделение значений в зависимости от классов * Постоянно придумывать идеи для новых признаков (где-то их фиксировать) **3. Предобработка данных** * Разделить данные на train и test * Зависит от выбранной модели * Масштабирование * Заполнить пропуски 1. Выкинуть эти данные 2. Заменять разными методами (медианы, средние значения, бизнес-логика, строить модели...) 3. Делать/не делать дополнительную фичу 4. Ничего не делать * Обработать выбросы 1. Выкинуть эти данные 2. Заменять разными методами (медианы, средние значения, бизнес-логика, строить модели...) 3. Делать/не делать дополнительную фичу 4. Ничего не делать * Генерация новых признаков 1. Категориальные признаки a. pd.get_dummies, OneHotEncoder b. Feature Encoding c. Target Encoding 2. Вещественные признаки a. Feature discretization b. Feature binarization * Оформить предобработку в класс/функции для воспроизводимости **4. Обучение и валидация моделей** * Обучить модель на базовых гиперпараметрах * Вручную настраивать гиперпарметры * Автоматизированный выбор гиперпараметров с кросс-валидацией GridSearchCV/RandomSearchCV * Следить за переобучением, если оно появляется, то искать лучший параметр регуляризации * Посчитать метрики **5. Воспроизводимость** * Зафиксировать результаты эксперимента * Сохранить модели * Сохранить версии библиотек **6. Вы и ваша модель идеальна! :)**

Ввод [75]:
```
!pip freeze >> requirements.txt
```

Ввод [ ]: