Deep Learning School

Школа глубокого обучения ФПМИ МФТИ

Домашнее задание. Продвинутый поток. Весна 2021

Это домашнее задание будет посвящено полноценному решению задачи машинного обучения.

Предсказание оттока пользователей (весна 2021)

Задача

Вам предстоит научиться моделировать отток клиентов телеком компании. Эта задача очень важна на практике и алгоритмы для ее решения используются в реальных телеком компаниях, ведь если мы знаем, что клиент собирается уйти от нас, то мы попытаться удержать его, предложив какие-то бонусы.

Первая часть. Исследование

```
1 # подключаем необходимые библиотеки
```

```
!pip install catboost
import pandas as pd
import numpy as np

from matplotlib import pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearchCV

from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.metrics import accuracy_score
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import confusion_matrix, precision_score, recall_score, classifica

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, RobustScaler, LabelEncoder, OneHotEnc
from sklearn.pipeline import make_pipeline, Pipeline

from sklearn.linear_model import LogisticRegression, LogisticRegressionCV


import lightgbm as lgb
from catboost import CatBoostClassifier
from google.colab import drive

drive.mount('/content/gdrive')
```

```
Requirement already satisfied: catboost in /usr/local/lib/python3.7/dist-packages (1
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from ca
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.7/dist-packa
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive
```

Загрузка и подготовка данных (2 балла)

```
# для работы локально нужно выставить флаг local = True
# для чтения данных с с Google диска local = False
local = False
```

```
# чтение данных
if local:
    train = pd.read_csv('./train.csv')
    test =  pd.read_csv('./test.csv')
```

```
else:
    # чтение с Google диска

    #train

    train = pd.read_csv('/content/gdrive/MyDrive/Colab_Notebooks/prediction of outflow/

    #test
    test = pd.read_csv('/content/gdrive/MyDrive/Colab_Notebooks/prediction of outflow/t

```

```
# Для вашего удобства списки с именами разных колонок

# Числовые признаки
num_cols = [
    'ClientPeriod',
    'MonthlySpending',
    'TotalSpent'
]

# Категориальные признаки
cat_cols = [
    'Sex',
    'IsSeniorCitizen',
    'HasPartner',
    'HasChild',
    'HasPhoneService',
    'HasMultiplePhoneNumbers',
    'HasInternetService',
    'HasOnlineSecurityService',
    'HasOnlineBackup',
    'HasDeviceProtection',
    'HasTechSupportAccess',
    'HasOnlineTV',
    'HasMovieSubscription',
    'HasContractPhone',
    'IsBillingPaperless',
    'PaymentMethod'
]

feature_cols = num_cols + cat_cols
target_col = 'Churn'
```

Смотрим на данные.

```
train.head(5)
```

	ClientPeriod	MonthlySpending	TotalSpent	Sex	IsSeniorCitizen	HasPartner	Ha
0	55	19.50	1026.35	Male	0	Yes	
1	72	25.85	1872.2	Male	0	Yes	
2	1	75.90	75.9	Male	0	No	

```
1 train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5282 entries, 0 to 5281
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   ClientPeriod              5282 non-null   int64  
 1   MonthlySpending           5282 non-null   float64
 2   TotalSpent                5282 non-null   object 
 3   Sex                       5282 non-null   object 
 4   IsSeniorCitizen           5282 non-null   int64  
 5   HasPartner                5282 non-null   object 
 6   HasChild                  5282 non-null   object 
 7   HasPhoneService           5282 non-null   object 
 8   HasMultiplePhoneNumbers   5282 non-null   object 
 9   HasInternetService        5282 non-null   object 
 10  HasOnlineSecurityService  5282 non-null   object 
 11  HasOnlineBackup           5282 non-null   object 
 12  HasDeviceProtection       5282 non-null   object 
 13  HasTechSupportAccess      5282 non-null   object 
 14  HasOnlineTV               5282 non-null   object 
 15  HasMovieSubscription      5282 non-null   object 
 16  HasContractPhone          5282 non-null   object 
 17  IsBillingPaperless        5282 non-null   object 
 18  PaymentMethod             5282 non-null   object 
 19  Churn                     5282 non-null   int64  
dtypes: float64(1), int64(3), object(16)
memory usage: 825.4+ KB
```

```
1 test.head(5)
```

	ClientPeriod	MonthlySpending	TotalSpent	Sex	IsSeniorCitizen	HasPartner	Ha

```
1 test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1761 entries, 0 to 1760
Data columns (total 19 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   ClientPeriod             1761 non-null   int64  
 1   MonthlySpending          1761 non-null   float64
 2   TotalSpent               1761 non-null   object 
 3   Sex                      1761 non-null   object 
 4   IsSeniorCitizen          1761 non-null   int64  
 5   HasPartner               1761 non-null   object 
 6   HasChild                 1761 non-null   object 
 7   HasPhoneService          1761 non-null   object 
 8   HasMultiplePhoneNumbers  1761 non-null   object 
 9   HasInternetService       1761 non-null   object 
 10  HasOnlineSecurityService 1761 non-null   object 
 11  HasOnlineBackup          1761 non-null   object 
 12  HasDeviceProtection      1761 non-null   object 
 13  HasTechSupportAccess     1761 non-null   object 
 14  HasOnlineTV              1761 non-null   object 
 15  HasMovieSubscription     1761 non-null   object 
 16  HasContractPhone         1761 non-null   object 
 17  IsBillingPaperless       1761 non-null   object 
 18  PaymentMethod            1761 non-null   object 
dtypes: float64(1), int64(2), object(16)
memory usage: 261.5+ KB
```

Вывод: Пропусков в данных нет.

Преобразование типов данных

Посмотрим количества уникальных **unique** значений различных признаков строкового типа.

```
1 train.describe(include=[np.object])
```

	TotalSpent	Sex	HasPartner	HasChild	HasPhoneService	HasMultiplePhoneNum
count	5282	5282	5282	5282	5282	
unique	4978	2	2	2	2	
top	20.2	Male	No	No	Yes	
freq	9	2655	2705	3676	4761	

Все признаки кроме TotalSpent переведем в категориальный тип. А TotalSpent в тип float64.

Смотрим на данные числового типа.

```
1 train.describe(include=[np.number])
```

	ClientPeriod	MonthlySpending	IsSeniorCitizen	Churn
count	5282.000000	5282.000000	5282.000000	5282.000000
mean	32.397009	64.924754	0.159409	0.262022
std	24.550326	30.176464	0.366092	0.439776
min	0.000000	18.250000	0.000000	0.000000
25%	9.000000	35.462500	0.000000	0.000000
50%	29.000000	70.400000	0.000000	0.000000
75%	55.000000	90.050000	0.000000	1.000000
max	72.000000	118.750000	1.000000	1.000000

SeniorCitizen переведем в категориальный тип.

```
1 # SeniorCitizen -> category
2 train['IsSeniorCitizen'] = train['IsSeniorCitizen'].astype('category')
3 test['IsSeniorCitizen'] = test['IsSeniorCitizen'].astype('category')
```

В строковом типе TotalSpent пропуски заменим на средние значения и преобразуем в float64

```
1 train["TotalSpent"] = pd.to_numeric(train["TotalSpent"], errors="coerce")
2 test["TotalSpent"] = pd.to_numeric(test["TotalSpent"], errors="coerce")
```

```
1 train['TotalSpent'] = train['TotalSpent'].replace(' ', train['TotalSpent'].mean())
2 train['TotalSpent'] = train['TotalSpent'].astype('float64')
3
4 test['TotalSpent'] = test['TotalSpent'].replace(' ',test['TotalSpent'].mean())
5 test['TotalSpent'] = test['TotalSpent'].astype('float64')
```

Все строковые переводим в категориальные.

```
1 # строковые типы переводим в категориальные
2 list_object_columns = list(train.dtypes[train.dtypes == object].index)
3
4 for col in list_object_columns:
```

```
    train[col] = train[col].astype('category')
    test[col] = test[col].astype('category')
```

Смотрим результат.

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5282 entries, 0 to 5281
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype   
---  ------                    --------------  -----   
 0   ClientPeriod              5282 non-null   int64   
 1   MonthlySpending           5282 non-null   float64 
 2   TotalSpent                5273 non-null   float64 
 3   Sex                       5282 non-null   category
 4   IsSeniorCitizen           5282 non-null   category
 5   HasPartner                5282 non-null   category
 6   HasChild                  5282 non-null   category
 7   HasPhoneService           5282 non-null   category
 8   HasMultiplePhoneNumbers   5282 non-null   category
 9   HasInternetService        5282 non-null   category
 10  HasOnlineSecurityService  5282 non-null   category
 11  HasOnlineBackup           5282 non-null   category
 12  HasDeviceProtection       5282 non-null   category
 13  HasTechSupportAccess      5282 non-null   category
 14  HasOnlineTV               5282 non-null   category
 15  HasMovieSubscription      5282 non-null   category
 16  HasContractPhone          5282 non-null   category
 17  IsBillingPaperless        5282 non-null   category
 18  PaymentMethod             5282 non-null   category
 19  Churn                     5282 non-null   int64   
dtypes: category(16), float64(2), int64(2)
memory usage: 249.4 KB
```

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1761 entries, 0 to 1760
Data columns (total 19 columns):
 #   Column                    Non-Null Count  Dtype   
---  ------                    --------------  -----   
 0   ClientPeriod              1761 non-null   int64   
 1   MonthlySpending           1761 non-null   float64 
 2   TotalSpent                1759 non-null   float64 
 3   Sex                       1761 non-null   category
 4   IsSeniorCitizen           1761 non-null   category
 5   HasPartner                1761 non-null   category
 6   HasChild                  1761 non-null   category
 7   HasPhoneService           1761 non-null   category
 8   HasMultiplePhoneNumbers   1761 non-null   category
 9   HasInternetService        1761 non-null   category
 10  HasOnlineSecurityService  1761 non-null   category
 11  HasOnlineBackup           1761 non-null   category
 12  HasDeviceProtection       1761 non-null   category
 13  HasTechSupportAccess      1761 non-null   category
 14  HasOnlineTV               1761 non-null   category
```

```
 15  HasMovieSubscription      1761 non-null   category
 16  HasContractPhone          1761 non-null   category
 17  IsBillingPaperless        1761 non-null   category
 18  PaymentMethod             1761 non-null   category
dtypes: category(16), float64(2), int64(1)
memory usage: 70.6 KB
```

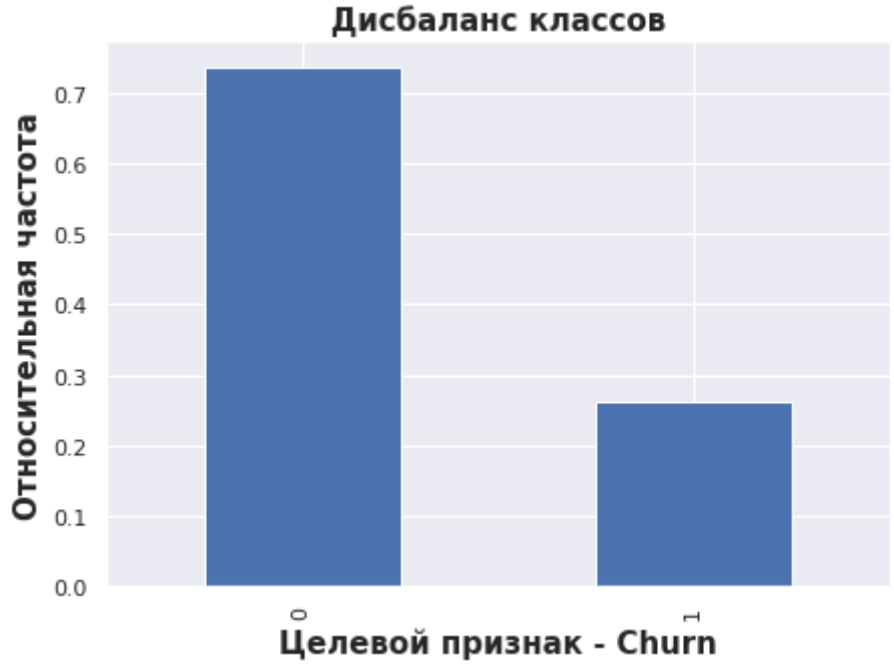
Вывод: Данные загружены и подготовлены к проведению EDA.

EDA Анализ данных (3 балла)

Рассмотрим датасет на наличие дисбаланса классов.

```
1 sns.set(rc={'figure.figsize':(7, 5)})
2 class_frequency = train['Churn'].value_counts(normalize=True)
3 class_frequency.plot(kind='bar')
4 plt.xlabel('Целевой признак - Churn',fontsize=15, weight = 'bold')
5 plt.ylabel('Относительная частота',fontsize=15, weight = 'bold')
6 plt.title("Дисбаланс классов",fontsize=15, weight = 'bold')
7
```

```
Text(0.5, 1.0, 'Дисбаланс классов')
```




```
1 print(class_frequency)
```

```
0    0.737978
1    0.262022
Name: Churn, dtype: float64
```

```
1 class_frequency[0]/class_frequency[1]
```

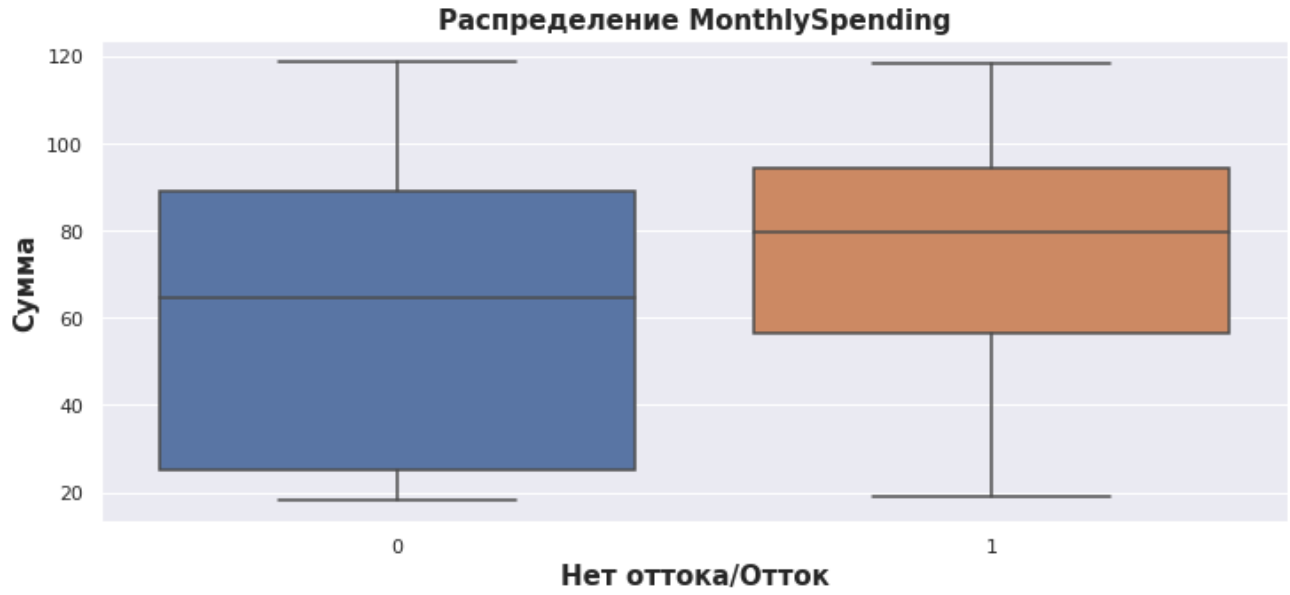
```
2.816473988439306
```

Вывод: Мы имеем выраженный дисбаланс классов в отношении 0 класса к 1 классу, как 3:1. Для повышения качетсва метрики при обучении моделей этот факт необходимо будет учесть.

Распределение численных признаков

MonthlySpending

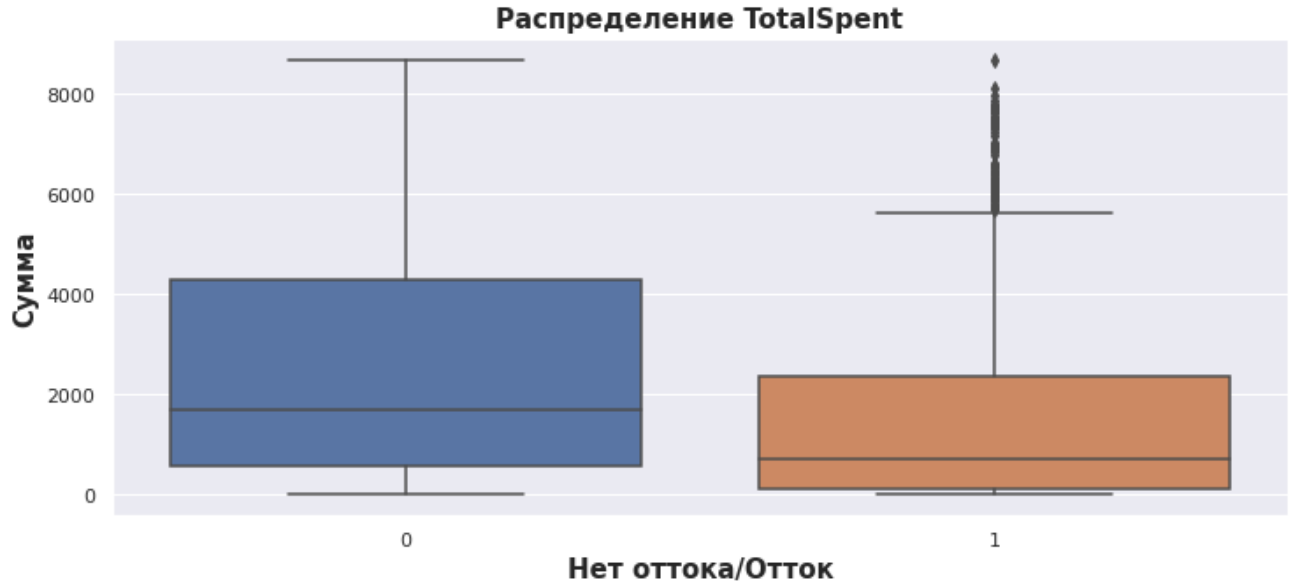
```
1 plt.figure(figsize=(12, 5))
2 ax = sns.boxplot(x='Churn', y = 'MonthlySpending', data=train);
3 plt.title("Распределение MonthlySpending", fontsize=15, weight = 'bold')
4 plt.ylabel('Сумма',fontsize=15, weight = 'bold')
5 plt.xlabel('Нет оттока/Отток',fontsize=15, weight = 'bold');
```



Вывод: Чем больше ежемесячные платежи, тем больше отток.

TotalSpent

```
1 plt.figure(figsize=(12, 5))
2 ax = sns.boxplot(x='Churn', y = 'TotalSpent', data=train);
3 plt.title("Распределение TotalSpent", fontsize=15, weight = 'bold')
4 plt.ylabel('Сумма',fontsize=15, weight = 'bold')
5 plt.xlabel('Нет оттока/Отток',fontsize=15, weight = 'bold');
```

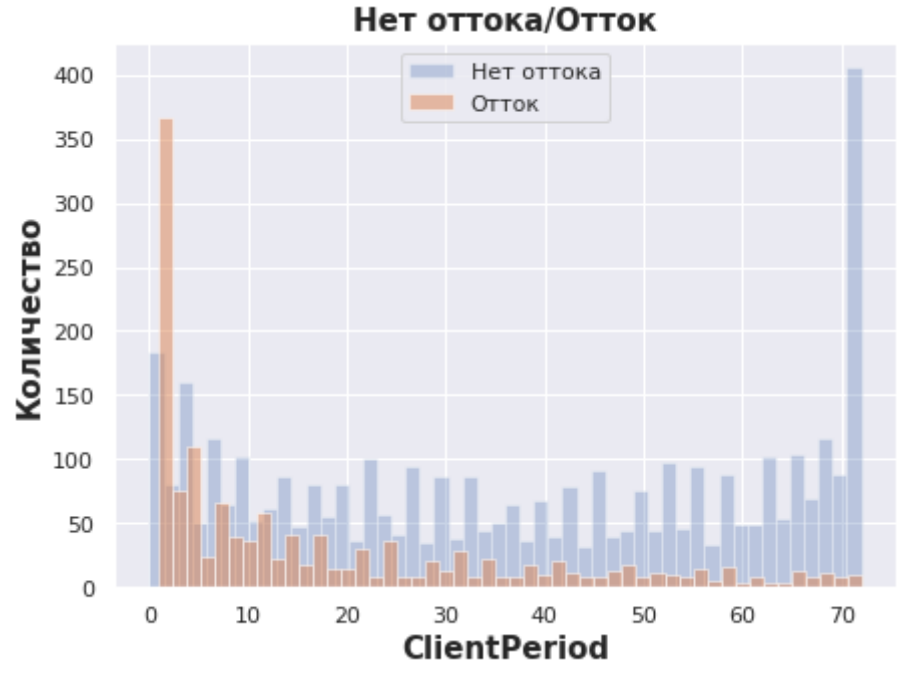


Вывод: Общая сумма платежа не успевает повлиять на отток, клиент уходит раньше, чем выплатит всю сумму. По целевому признаку есть выбросы.

ClientPeriod

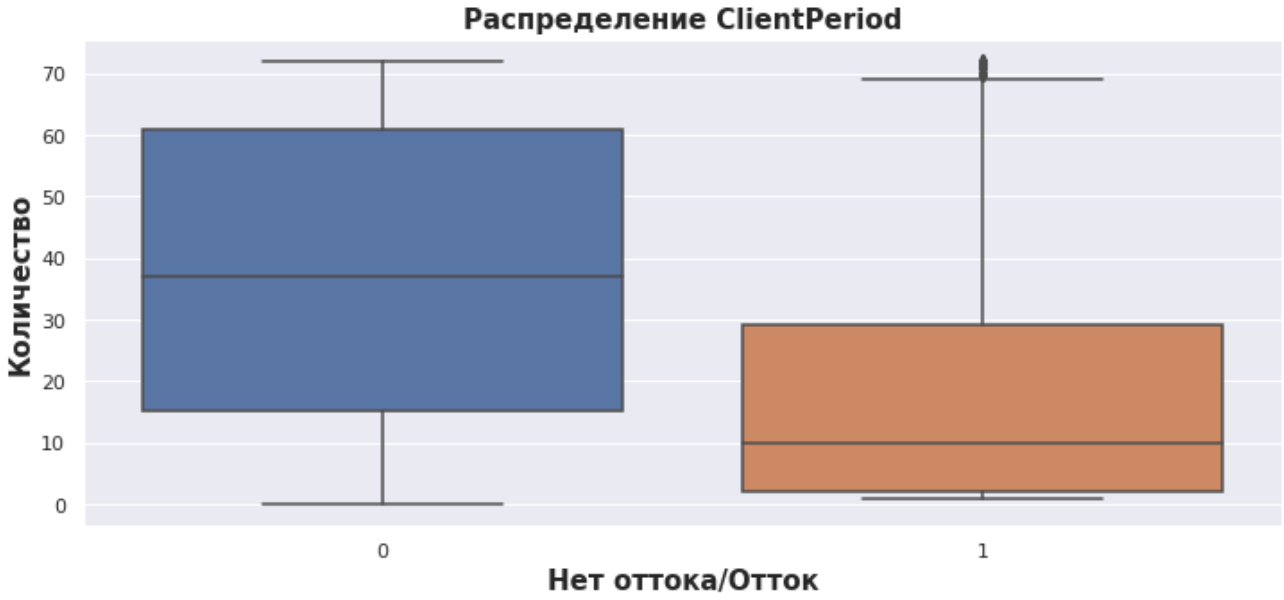
```
1 ax = train[train['Churn']==0]['ClientPeriod'].hist(bins=50, alpha=0.3)
2 train[train['Churn']==1]['ClientPeriod'].hist(bins=50, alpha=0.5)
3 ax.legend(['Нет оттока', 'Отток'])
4
5 plt.title("Нет оттока/Отток",fontsize=15, weight = 'bold');
6 plt.xlabel('ClientPeriod', fontsize=15, weight = 'bold')
7 plt.ylabel('Количество', fontsize=15, weight = 'bold')
```

```
Text(0, 0.5, 'Количество')
```




```
1 plt.figure(figsize=(12, 5))
2 ax = sns.boxplot(x='Churn', y = 'ClientPeriod', data=train);
3 plt.title("Распределение ClientPeriod", fontsize=15, weight = 'bold')
```

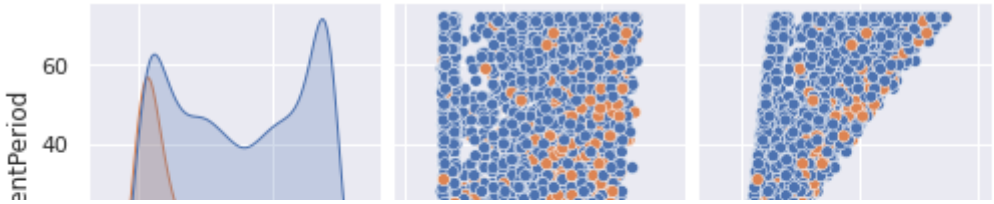
```
4 plt.ylabel('Количество',fontsize=15, weight = 'bold')
5 plt.xlabel('Нет оттока/Отток',fontsize=15, weight = 'bold');
```



Вывод: Чем меньше срок взаимодействия клиента и кампании тем больше отток. По целевому признаку на трэйне есть выбросы.

Все три признака вместе:

```
1 sns.pairplot(train[num_cols + [target_col]], hue=target_col);
```



Вывод: Подозрительно хорошо коррелируют пары (ClientPeriod, TotalSpent) и (MonthlySpending, TotalSpent)

Проверка на мультиколлинеарность признаков

Для числовых переменных можно построить матрицу корреляции Пирсона.

```
1 train[['ClientPeriod','MonthlySpending', 'TotalSpent']].corr()
```

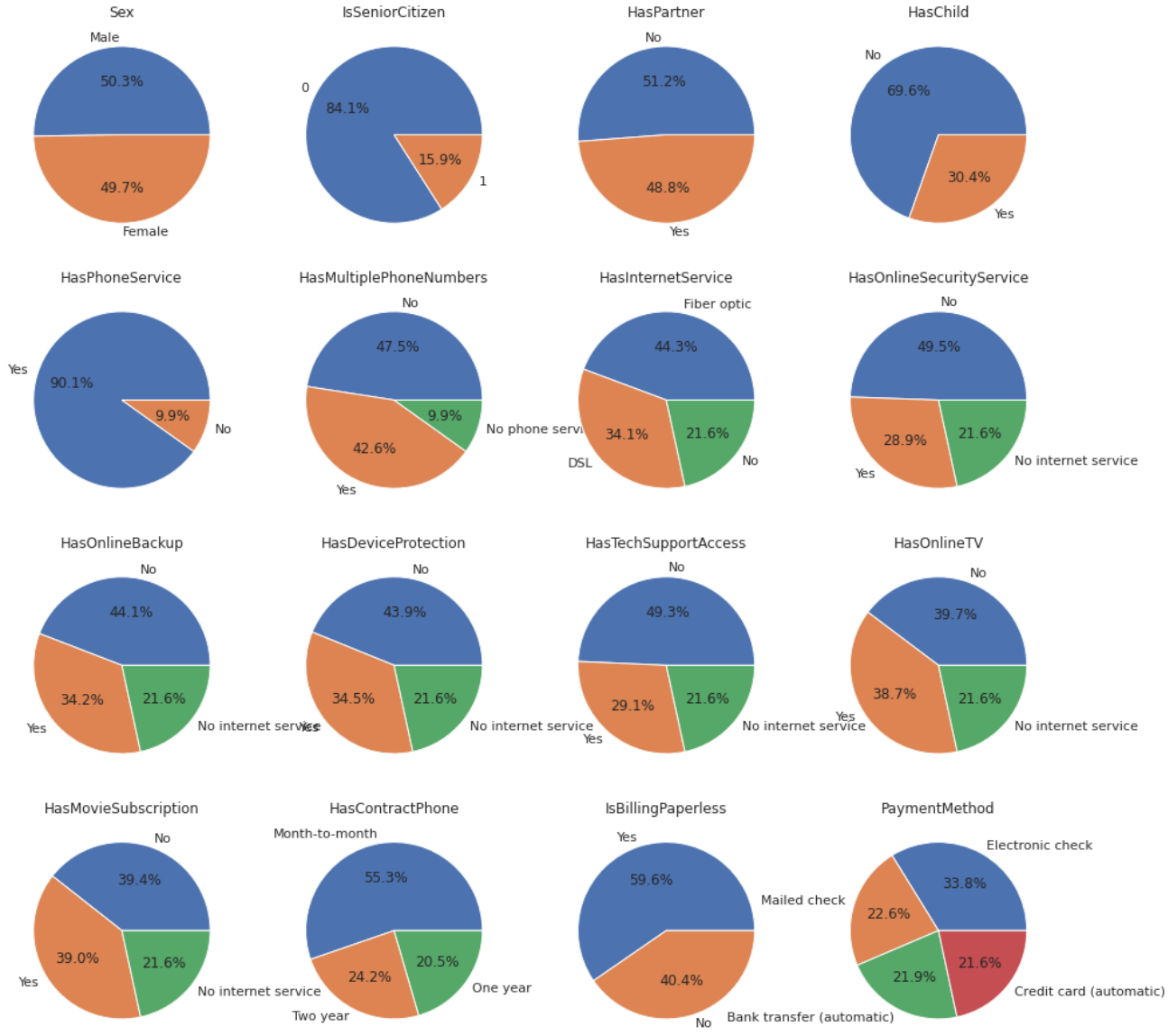
	ClientPeriod	MonthlySpending	TotalSpent
ClientPeriod	1.000000	0.249414	0.826332
MonthlySpending	0.249414	1.000000	0.652034
TotalSpent	0.826332	0.652034	1.000000

Корреляция между TotalSpent и ClientPeriod равна 0.82, также между TotalSpent и MonthlySpending корреляция пирсона равна 0.65. Всё это может негативно сказаться на моделях. Перед обучением модели нужно будет удалить потенциально опасный признак TotalSpent.

Относительные частоты категориальных признаков

```
1 def category_value_count(data):
2     fig, axs = plt.subplots(4, 4, figsize=(16, 16))
3     for i, col in enumerate(cat_cols):
4         axs[i // 4][i % 4].pie(
5             data[col].value_counts(normalize=True),
6             labels=data[col].value_counts().index,
7             autopct='%1.1f%%'
8         )
9         axs[i // 4][i % 4].set_title(col)
```

```
1 category_value_count(train)
```

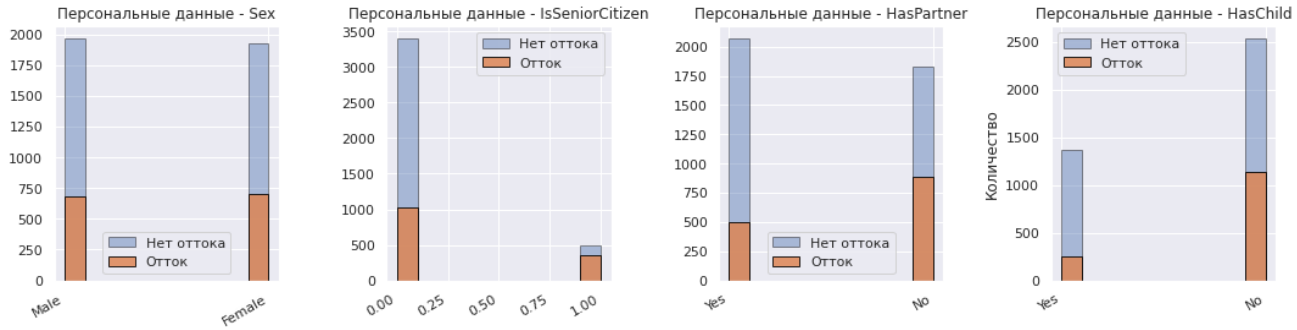


Влияние различных значений категориальных признаков на отток (распределение целевой переменной по признакам)

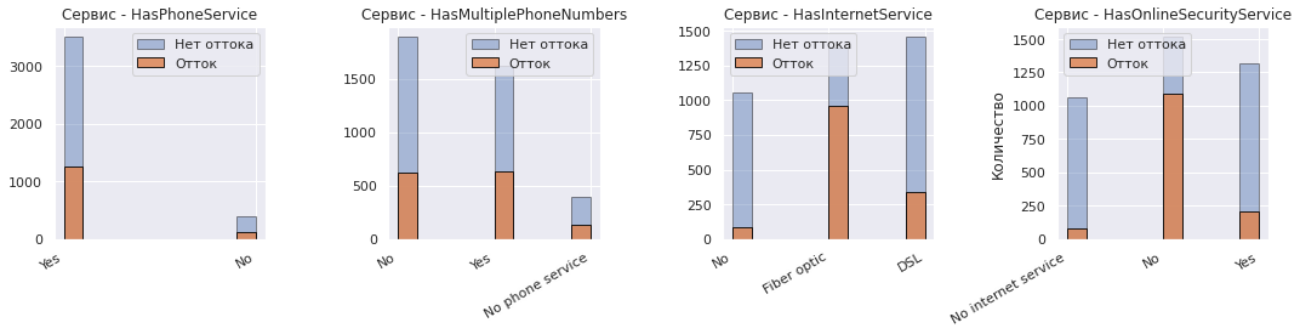
```
1 def plot_hist_groups(data, cols, n_cols, title='сервис'):
2
3     fig, ax = plt.subplots(nrows=1, ncols=n_cols, figsize = (15,4))
4     for i, col in enumerate(cols):
5
6         ax[i].hist(data[data['Churn'] == 0][col], ec='black', alpha=0.42)
```

```
        ax[i].hist(data[data['Churn'] == 1][col], ec='black', alpha=0.84)
        ax[i].set_title(f"{title} {col}")
        ax[i].legend(['Нет оттока', 'Отток'])

    plt.ylabel('Количество')
    fig.autofmt_xdate()
    plt.tight_layout()
    plt.show()
```

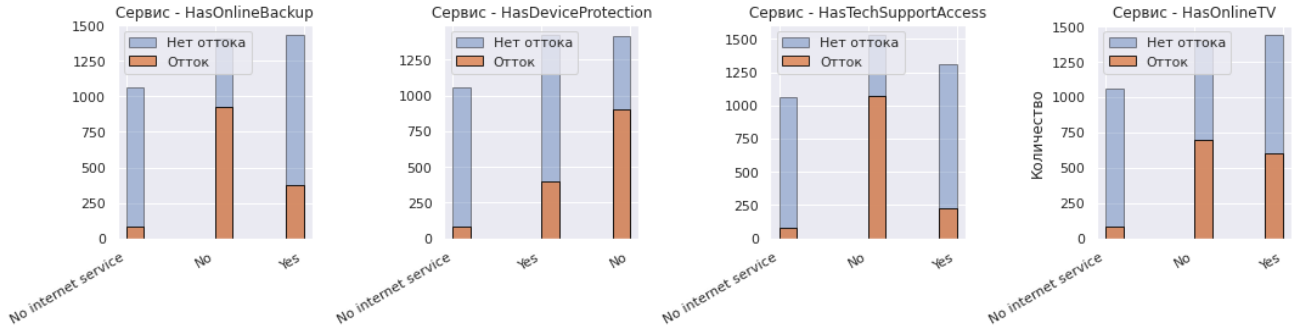
```
cols = ['Sex', 'IsSeniorCitizen', 'HasPartner', 'HasChild']
plot_hist_groups(train, cols, 4, 'Персональные данные -')
```



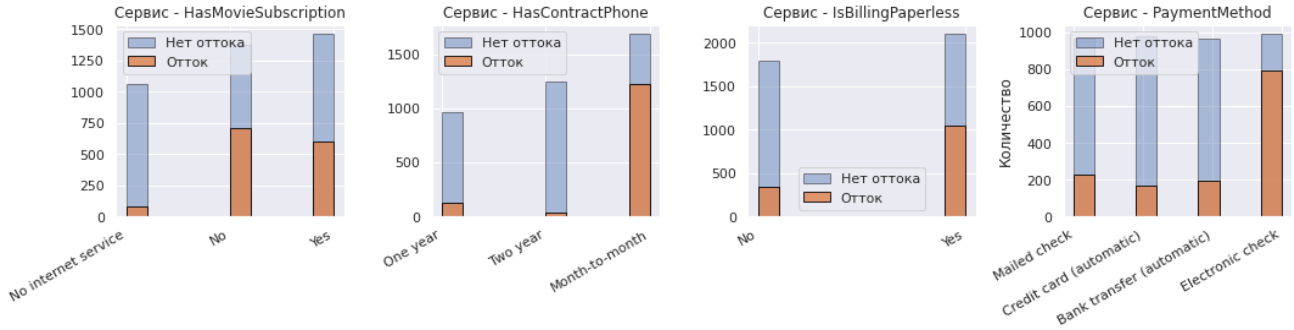

```
cols = ['HasPhoneService', 'HasMultiplePhoneNumbers', 'HasInternetService', 'HasOnlineS
plot_hist_groups(train, cols, 4, 'Сервис -')
```




```
cols = ['HasOnlineBackup', 'HasDeviceProtection', 'HasTechSupportAccess', 'HasOnlineTV'
plot_hist_groups(train, cols, 4, 'Сервис -')
```




```
cols = ['HasMovieSubscription', 'HasContractPhone', 'IsBillingPaperless', 'PaymentMetho
plot_hist_groups(train, cols, 4, 'Сервис -')
```



Выводы:

Не влияют на отток следующие признаки:

- Sex, HasMultiplePhoneNumbers, HasOnlineTV, HasMovieSubscription

Влияют на отток:

- PaymentMethod (видим отток при значении Electronic check)
- HasContractPhone (видим отток при значении Month-to-month)
- HasOnlineBackup, HasDeviceProtection, HasTechSupportAccess, HasOnlineSecurityService, HasInternetService, HasInternetService (видим отток при неподключении сервиса, значение - No)

Нелинейный зависимости

Нелинейные зависимости попробуем выявить через Phik (ϕk)

https://towardsdatascience.com/phik-k-get-familiar-with-the-latest-correlation-coefficient-9ba0032b37e7

Данный критерий работает с любыми типами переменных, включая категориальные. И основан на критерии Хи-квадрат.

Цитирую документацию:

"Phi_K is a new and practical correlation coefficient based on several refinements to Pearson's hypothesis test of independence of two variables.

The combined features of Phi_K form an advantage over existing coefficients. First, it works consistently between categorical, ordinal and interval variables. Second, it captures non-linear dependency. Third, it reverts to the Pearson correlation coefficient in case of a bi-variate normal input distribution. These are useful features when studying the correlation matrix of variables with mixed types.

.....

The calculation of correlation coefficients between paired data variables is a standard tool of analysis for every data analyst. Pearson's correlation coefficient is a de facto standard in most fields, but by construction only works for interval variables (sometimes called continuous variables). Pearson is unsuitable for data sets with mixed variable types, e.g. where some variables are ordinal or categorical".

```
1 # Библиотека для анализа нелинейных зависимостей
2 !pip3 install phik
3 import phik
4 from phik.report import plot_correlation_matrix
5 from phik import report
```

```
Requirement already satisfied: phik in /usr/local/lib/python3.7/dist-packages (0.12.0
Requirement already satisfied: matplotlib>=2.2.3 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: pandas>=0.25.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: scipy>=1.5.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from cy
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages
```

```
1 phik_overview = train.phik_matrix()
2 phik_overview.round(2)
3 phik_overview['Churn'].sort_values(ascending=False).to_frame().style.background_gradien
```

```
interval columns not set, guessing: ['ClientPeriod', 'MonthlySpending', 'TotalSpent',
```

	Churn
Churn	1.000000
ClientPeriod	0.469423
PaymentMethod	0.442566
MonthlySpending	0.360202
IsBillingPaperless	0.298737
TotalSpent	0.280625
HasChild	0.252610
HasContractPhone	0.250575
IsSeniorCitizen	0.242130
HasPartner	0.229302
HasOnlineSecurityService	0.217454
HasTechSupportAccess	0.208887
HasInternetService	0.192010
HasOnlineBackup	0.178838

Вывод: Наиболее значимыми и влияющими на отток признаками будут - ClientPeriod, PaymentMethod, MonthlySpending и IsBillingPaperless

Данный факт можно будет проверить по значению feature_importance для бустинговых моделей.

Применение линейных моделей (3 балла)

```
# собирать данные о моделях будем в список
result_list = []
```

Убираем TotalSpent для исключения ошибок связанных с коллиенарностью.

```
num_cols = ['ClientPeriod', 'MonthlySpending','TotalSpent']

df_train = train.copy()
if 'TotalSpent' in df_train.columns:
    del df_train['TotalSpent']
    num_cols = ['ClientPeriod', 'MonthlySpending']
```

Ранее мы выяснили, что на тестовой выборке есть выбросы по признаку ClientPeriod. Этот признак имеет большую важность при обучении.Уберем эти выбросы, так как они негативно скажуться на линейной модели.

```
# функция очищает датасет от выбросов, отбрасывая объекты за пределами межквартильного

def frame_irq(df, column_list=[], query=False, koeff=1.5):
    if len(column_list)==0:
        column_list = list(df.columns)
    for column in column_list:
```

```
    for column in column_list:
        q25 = df[column].quantile(0.25)
        q75 = df[column].quantile(0.75)
        minimum = df[column].min()
        maximum = df[column].max()
        irq = np.abs(q75 - q25)
        left = q25 - koeff*irq
        right = q75 + koeff*irq
        if query:
            sql_sentence='@left <= '+ column + ' and '+ column + ' <= @right'
            df = df.query(sql_sentence)
        else:
            df_tmp = df[(df[column]>left)&(df[column]<right)]
            if len(df_tmp) != 0:
                df=df_tmp.copy()
        pass
    return df
```

```
frames=[frame_irq(df_train.query('Churn==1'),['ClientPeriod']),df_train.query('Churn==0
df_train_irq15 = pd.concat(frames)
df_train_irq15.shape
```

```
(5264, 19)
```

```
df_train.shape
```

```
(5282, 19)
```

```
X = df_train_irq15.drop('Churn', axis=1)
y = df_train_irq15['Churn']
```

```
X_train, X_valid, y_train, y_valid = train_test_split (
                                X, y, test_size=0.25, random_state=42)
```

Создаем Pipeline для орбработки категориальных признаков для логистической регрессии.

```
# pipeline для LogisticRegression

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), num_cols),
        ('cat', OneHotEncoder(), cat_cols)])

pipe = Pipeline(steps=[('preprocessor', preprocessor),
                       ('classifier', LogisticRegression(random_state=42, class_weight =


```

Ищем лучшие параметры на сетке

```
grid = {"classifier__C": [100, 10, 1, 0.1, 0.01, 0.001], #Inverse of regularization str
        "classifier__penalty":['l2'],
        "preprocessor__cat__drop":['first']
        }
gcv = GridSearchCV(pipe, grid, cv=5, scoring='roc_auc', refit=True)
grid_result = gcv.fit(X_train, y_train)
```

```
print('best params:', grid_result.best_params_)
print('best score:', grid_result.best_score_)
```

```
best params: {'classifier__C': 1, 'classifier__penalty': 'l2', 'preprocessor__cat__dr
best score: 0.8489599760767522
```

```
lg_best_estimator = gcv.best_estimator_
```

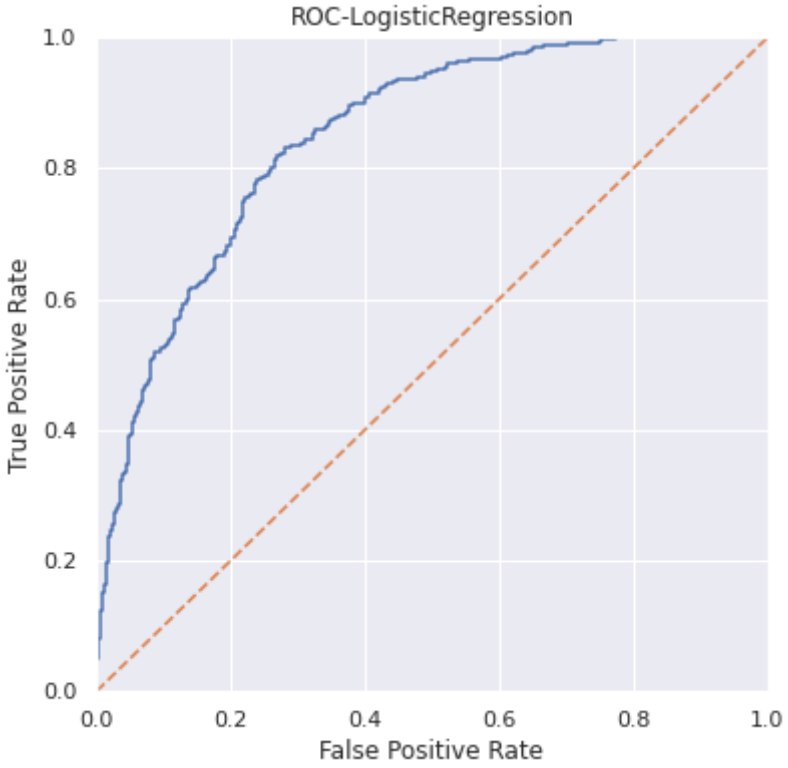
```
y_pred = lg_best_estimator.predict_proba(X_valid)[:, 1]
y_bin = (y_pred >= 0.5)*1
```

```
def plot_ROC(y_valid, y_pred, model=''):

    fpr, tpr, thresholds = roc_curve(y_valid, y_pred)
    plt.figure(figsize=(6, 6))
    plt.plot(fpr, tpr, linestyle='-')
    plt.plot([0, 1], [0, 1], linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.0])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC-'+ model)

    plt.show()
```

```
auc = roc_auc_score(y_valid, y_pred)
print("AUC:", auc)
print("")
print(classification_report(y_valid, y_bin))
```

```
AUC: 0.8486374586184224

              precision    recall  f1-score   support

           0       0.92      0.72      0.80       949
           1       0.53      0.83      0.65       367

    accuracy                           0.75      1316
   macro avg       0.72      0.77      0.73      1316
weighted avg       0.81      0.75      0.76      1316
```

```
plot_ROC(y_valid, y_pred, 'LogisticRegression')
```



Запишем посылку для соревнования на [kaggle]

```
#test = pd.read_csv('./test.csv')
X_test = test.copy()
if 'TotalSpent' in X_test.columns:
    del X_test['TotalSpent']

submission = pd.read_csv('/content/gdrive/MyDrive/Colab_Notebooks/prediction of outflow
submission['Churn'] = lg_best_estimator.predict_proba(X_test)[:, 1]
submission.to_csv('/content/gdrive/MyDrive/Colab_Notebooks/prediction of outflow/my_sub
```

```
# добавляем данные по модели
result_list.append({ 'Model':'LogisticRegression','roc_auc_score':round(auc,5),'kaggle
result_list
```

```
[{'Model': 'LogisticRegression',
  'kaggle auc': 0.84469,
  'roc_auc_score': 0.84864}]
```

Вывод Для линейной модели LogisticRegression удалось достичь качества метрики на трайн выборке AUC = 0.84864

kaggle AUC = 0.84469

Применение градиентного бустинга (2 балла)

CatBoostClassifier

Для catboost также возьмем сет очищенный от выбросов.

```
num_cols = ['ClientPeriod', 'MonthlySpending','TotalSpent']

df_train = df_train_irq15.copy() #train.copy()

if 'TotalSpent' in df_train.columns:
    del df_train['TotalSpent']
    num_cols = ['ClientPeriod', 'MonthlySpending']

X = df_train.drop('Churn', axis=1)
y = df_train['Churn']
```

CatBoostClassifier не требует технологии OHE для работы с категориальными данными. Достаточно просто указать категориальные колонки.

```
# получим список категориальных признаков
def get_category_features(df):
    category_cols = list(df.dtypes[df.dtypes == 'category'].index)
    cat_features_names = [col for col in df.columns if col in (category_cols)]
    cat_features = [df.columns.get_loc(col) for col in cat_features_names]
    return category_cols, cat_features_names, cat_features

cat_features,_,_ = get_category_features(X_train)
print(cat_features)
```

```
['Sex', 'IsSeniorCitizen', 'HasPartner', 'HasChild', 'HasPhoneService', 'HasMultipleP
```

Протестируем CatBoostClassifier со стандартными параметрами.

```
X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=0.8, random_stat
```

```
cat_model = CatBoostClassifier(cat_features=cat_features, random_seed=42, silent= True,
                               eval_metric='AUC:hints=skip_train~false', loss_function='

```

```
cat_model.fit(
   X_train, y_train,
   cat_features=cat_features,
   eval_set=(X_valid, y_valid),
   use_best_model=True,
   logging_level='Silent',
   plot=True
)
```

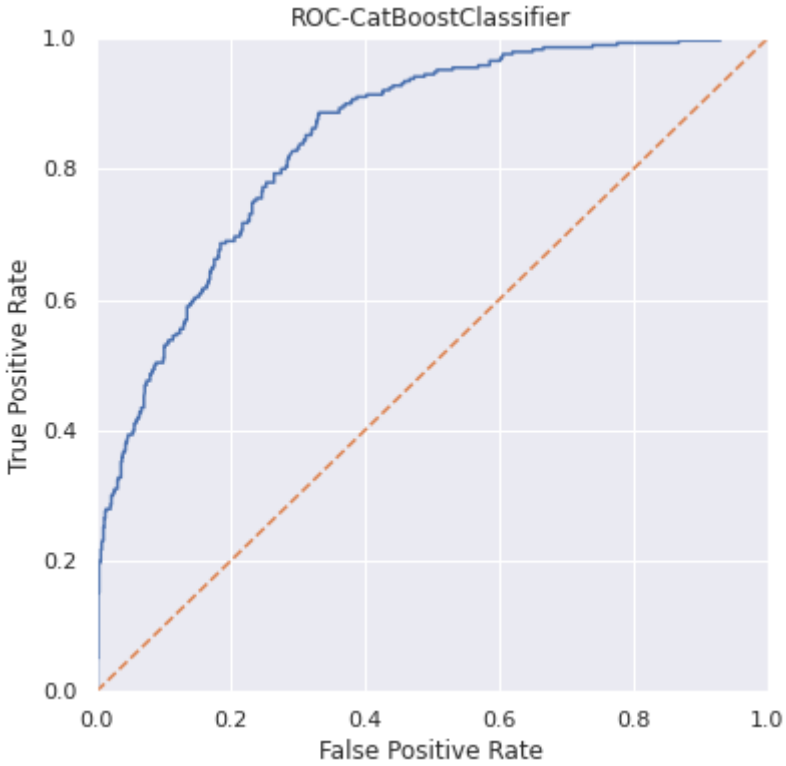
Смотрим результат на валидационной выборке.

```
y_pred = cat_model.predict_proba(X_valid)[:, 1]
y_bin = (y_pred >= 0.5)*1
```

```
auc = roc_auc_score(y_valid, y_pred)
print("AUC:", auc)
print("")
print(classification_report(y_valid, y_bin))
```

```
AUC: 0.8471663578433588

              precision    recall  f1-score   support

           0       0.83      0.89      0.86       763
           1       0.66      0.53      0.59       290

    accuracy                           0.79      1053
   macro avg       0.74      0.71      0.72      1053
weighted avg       0.78      0.79      0.79      1053
```

```
plot_ROC(y_valid, y_pred, 'CatBoostClassifier')
```



Результат с параметрами из коробки на тренировочной выборке оказался хуже чем у логистической регрессии. AUC = 0.8472

Попробуем поиск по сетке параметров и сбалансируем классы.

```
classes = np.unique(y_train)
weights = compute_class_weight(class_weight='balanced', classes=classes, y=y_train)
class_weights = dict(zip(classes, weights))
class_weights
```

```
{0: 0.6716108452950558, 1: 1.9567843866171004}
```

```
param_grid = {
     'learning_rate': [0.05, 0.1],
     'depth': [2,4],
     'l2_leaf_reg': [3,6]
}
```

```
%%time
cat_model = CatBoostClassifier(class_weights = class_weights, cat_features=cat_features
                               eval_metric='AUC:hints=skip_train~false', loss_function='

grid_search_result = cat_model.grid_search(param_grid, X=X_train, y=y_train, plot=False
                                           stratified = True, refit=True)
```

```
'%%time\ncat_model = CatBoostClassifier(class_weights = class_weights, cat_features=
cat_features, random_seed=42,                                  eval_metric='AUC:hints=s
kip_train~false', loss_function='Logloss', verbose=200)\n \ngrid_search_result = cat
 model.grid_search(param_grid, X=X_train, y=y_train, plot=False, cv = 5,
```

```
print(grid_search_result['params'])
```

```
# фиксируем лучший результат по auc на kaggle
cat_best_params = {'depth': 3, 'l2_leaf_reg': 5, 'learning_rate': 0.1}
```

```
cat_model = CatBoostClassifier(class_weights=class_weights, cat_features=cat_features,
                               random_seed=42, **cat_best_params, silent= True, eval_met
```

```
cat_model.fit(
   X_train, y_train,
   cat_features=cat_features,
   eval_set=(X_valid, y_valid),
   use_best_model=True,
   logging_level='Silent',
   plot=True
)
```

```
Custom logger is already specified. Specify more than one logger at same time is not
```

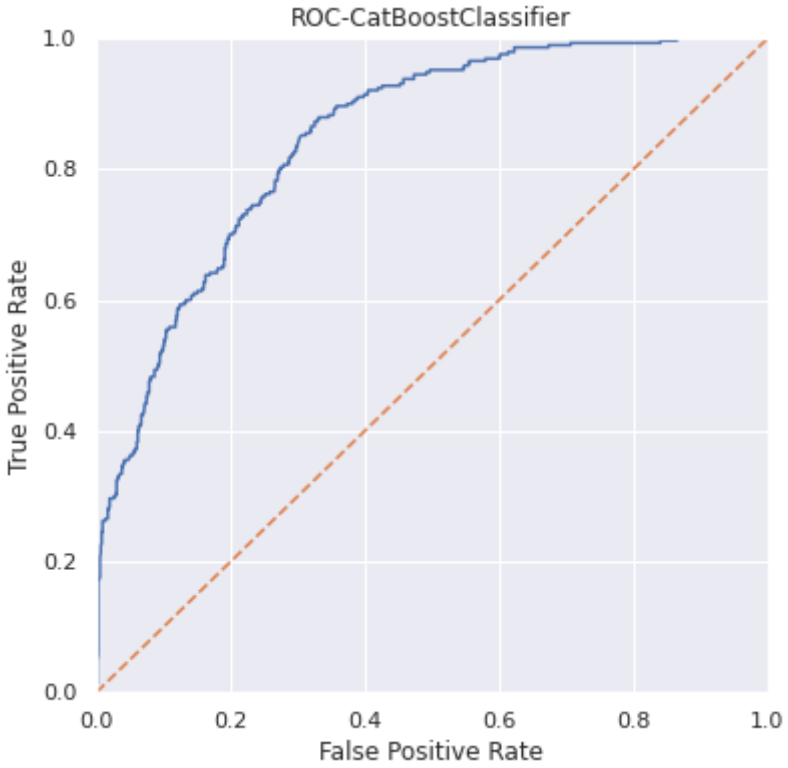
```
y_pred = cat_model.predict_proba(X_valid)[:, 1]
y_bin = (y_pred >= 0.5)*1
```

```
auc = roc_auc_score(y_valid, y_pred)
print("AUC:", auc)
print("")
print(classification_report(y_valid, y_bin))
```

```
4 print(classification_report(y_valid, y_bin))
```

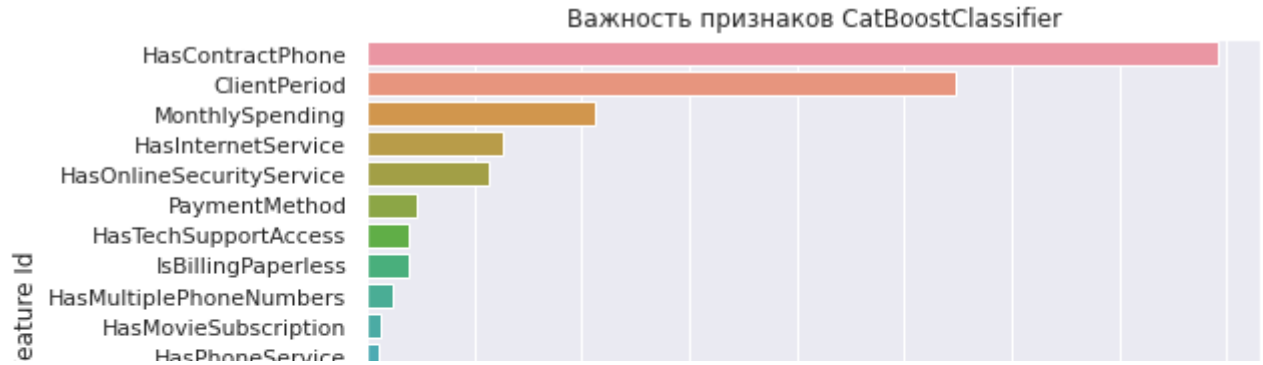
```
AUC: 0.8490396348352691

              precision    recall  f1-score   support

           0       0.91      0.70      0.80       763
           1       0.52      0.83      0.63       290

    accuracy                           0.74      1053
   macro avg       0.71      0.77      0.72      1053
weighted avg       0.80      0.74      0.75      1053
```

```
1 plot_ROC(y_valid, y_pred, 'CatBoostClassifier')
```



Важность признаков

```
1 fi = cat_model.get_feature_importance(prettified=True)
2
3 plt.figure(figsize=(8, 5));
4 sns.barplot(x="Importances", y="Feature Id", data=fi);
5 plt.title('Важность признаков CatBoostClassifier');
```



Важность топ 5 признаков совпадает с важностью признаков оцененной в ходе EDA.

Запишем посылку для соревнования на [kaggle]


```
#test = pd.read_csv('./test.csv')
X_test = test.copy()
if 'TotalSpent' in X_test.columns:
    del X_test['TotalSpent']

submission = pd.read_csv('/content/gdrive/MyDrive/Colab_Notebooks/prediction of outflow
submission['Churn'] = cat_model.predict_proba(X_test)[:, 1]
submission.to_csv('/content/gdrive/MyDrive/Colab_Notebooks/prediction of outflow/my_sub
```

```
# добавляем данные по модели
result_list.append({ 'Model':'CatBoostClassifier','roc_auc_score':round(auc,5),'kaggle
result_list
```

```
[{'Model': 'LogisticRegression',
  'kaggle auc': 0.84469,
  'roc_auc_score': 0.84864},
 {'Model': 'CatBoostClassifier',
  'kaggle auc': 0.85351,
  'roc_auc_score': 0.84904}]
```

Вывод Для CatBoostClassifier удалось достичь качества метрики на трайн выборке ROC_AUC = 0.84904

kaggle AUC = 0.0.84864

LGBMClassifier

Посмотрим ещё одну модель градиентного бустинга LGBMClassifier. Эта модель не требует предобработки категориальных признаков.

```
df_train = df_train_irq15.copy() #train.copy()

if 'TotalSpent' in df_train.columns:
    del df_train['TotalSpent']
```

```
    num_cols = ['ClientPeriod', 'MonthlySpending']

X = df_train.drop('Churn', axis=1)
y = df_train['Churn']
```

```
X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=0.8, random_stat
```

```
%%time
lg = lgb.LGBMClassifier(silent=True, class_weight = 'balanced')

param_list = {"max_depth": [1, 2, 3],
              "num_leaves":[2, 4, 6],
              "learning_rate" : [0.05,0.1],
              "boosting_type": ['gbdt'],
              "n_estimators": [90,100,120],
              "reg_lambda" : [1,5,10]
}

my_cv = StratifiedKFold(n_splits=5).split(X_train, y_train)

grid_search = GridSearchCV(lg, n_jobs = -1, param_grid = param_list, cv = my_cv, \
                           scoring = "roc_auc", verbose = 5)
grid_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 162 candidates, totalling 810 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed:    2.4s
[Parallel(n_jobs=-1)]: Done  68 tasks      | elapsed:    7.1s
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:   15.2s
[Parallel(n_jobs=-1)]: Done 284 tasks      | elapsed:   27.3s
[Parallel(n_jobs=-1)]: Done 446 tasks      | elapsed:   43.3s
[Parallel(n_jobs=-1)]: Done 644 tasks      | elapsed:  1.0min
CPU times: user 11.3 s, sys: 258 ms, total: 11.6 s
Wall time: 1min 18s
[Parallel(n_jobs=-1)]: Done 810 out of 810 | elapsed:  1.3min finished
```

```
best_params_lgb = grid_search.best_params_
```

```
best_params_lgb
```

```
{'boosting_type': 'gbdt',
 'learning_rate': 0.1,
 'max_depth': 2,
 'n_estimators': 100,
 'num_leaves': 4,
 'reg_lambda': 10}
```

```
lgb_model = lgb.LGBMClassifier(**best_params_lgb, silent=True, class_weight = 'balanced

lgb_model.fit(X_train, y_train, eval_set=(X_valid, y_valid), verbose = False)
```

```
LGBMClassifier(boosting_type='gbdt', class_weight='balanced',
               colsample_bytree=1.0, importance_type='split', learning_rate=0.1,
               max_depth=2, min_child_samples=20, min_child_weight=0.001,
```

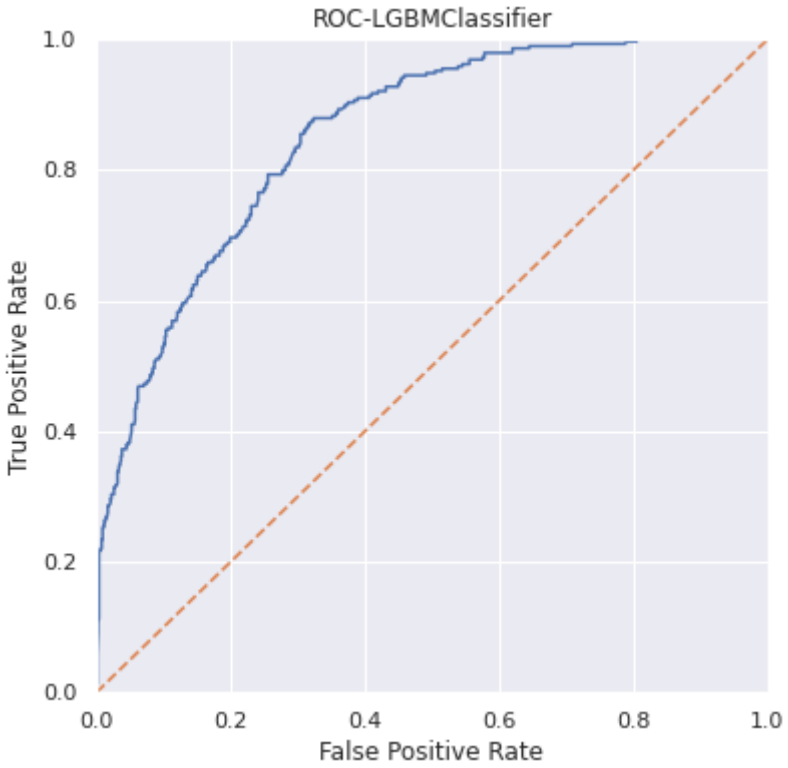
```
                   min_split_gain=0.0, n_estimators=100, n_jobs=-1, num_leaves=4,
                   objective=None, random_state=42, reg_alpha=0.0, reg_lambda=10,
                   silent=True, subsample=1.0, subsample_for_bin=200000,
                   subsample_freq=0)
```

```
1 y_pred = lgb_model.predict_proba(X_valid)[:, 1]
2 y_bin = (y_pred >= 0.5)*1
```

```
1 auc = roc_auc_score(y_valid, y_pred)
2 print("AUC:", auc)
3 print("")
4 print(classification_report(y_valid, y_bin))
```

```
    AUC: 0.8532720206083065

                  precision    recall  f1-score   support

               0       0.92      0.70      0.79       763
               1       0.51      0.83      0.64       290

        accuracy                           0.74      1053
       macro avg       0.72      0.77      0.71      1053
    weighted avg       0.81      0.74      0.75      1053
```

```
1 plot_ROC(y_valid, y_pred, 'LGBMClassifier')
```



Важность признаков LGBMClassifier

```
1 ax0 = lgb.plot_importance(lgb_model, color='b', title='Важность признаков LGBMClassifie
2                           ylabel='Признак', figsize=(7,5))
```

Важность топ 5 признаков совпадает с важностью оцененной в ходе EDA. Но по сравнению с catboost более важным стал признак ClientPeriod.

Запишем посылку для соревнования на [kaggle]

```
#test = pd.read_csv('./test.csv')
X_test = test.copy()
if 'TotalSpent' in X_test.columns:
    del X_test['TotalSpent']

submission = pd.read_csv('/content/gdrive/MyDrive/Colab_Notebooks/prediction of outflow
submission['Churn'] = lgb_model.predict_proba(X_test)[:, 1]
submission.to_csv('/content/gdrive/MyDrive/Colab_Notebooks/prediction of outflow/my_sub
```

```
# добавляем данные по модели
result_list.append({ 'Model':'LGBMClassifier','roc_auc_score':round(auc,5),'kaggle auc'
```

Вывод Для LGBMClassifier удалось достичь качества метрики на трайн выборке ROC_AUC = 0.8523

kaggle AUC = 0.84928

```
# соберем итоговую табличку
result = pd.DataFrame(result_list)
result = result.sort_values(by=['kaggle auc'], ascending = False).reset_index().drop('i
```

Выводы

Для прогнозированния оттока клиентов было проведено исследование EDA. Исследование показало:

Не влияют на отток следующие признаки:

- Sex, HasMultiplePhoneNumbers, HasOnlineTV, HasMovieSubscription

Влияют на отток:

- MonthlySpending (чем больше платежи, тем больше отток)
- ClientPeriod (больший отток в первые периоды)
- PaymentMethod (видим отток при значении Electronic check)
- HasContractPhone (видим отток при значении Month-to-month)
- HasOnlineBackup, HasDeviceProtection, HasTechSupportAccess, HasOnlineSecurityService, HasInternetService, HasInternetService (видим отток при неподключении сервиса, значение - No)

В дальнейшем важность признаков по EDA была подтверждена при помощи feature_importance бустинговыми моделями.

Были построены три модели машинного обучения: линейная - LogisticRegression, и две бустинговых LGBMClassifier и CatBoostClassifier. Результат обучения и тестирования на kaggle сведен в итоговую таблицу.

```
1 result
```

Лучшие результаты показал бустинг **CatBoostClassifier** с метрикой на kaggle **AUC = 0.85351**

Предсказания

```
1 best_model = cat_model
```

```
1 #test = pd.read_csv('./test.csv')
2 X_test = test.copy()
3 if 'TotalSpent' in X_test.columns:
4     del X_test['TotalSpent']
5
6 submission = pd.read_csv('/content/gdrive/MyDrive/Colab_Notebooks/prediction of outflow
7 submission['Churn'] = cat_model.predict_proba(X_test)[:, 1]
8 submission.to_csv('/content/gdrive/MyDrive/Colab_Notebooks/prediction of outflow/my_sub
```

Team name Daniel_Matveev

Kaggle (5 баллов)

Как выставить баллы:

1) 1 >= roc auc > 0.84 это 5 баллов

2) 0.84 >= roc auc > 0.7 это 3 балла

3) 0.7 >= roc auc > 0.6 это 1 балл

4) 0.6 >= roc auc это 0 баллов

Для выполнения задания необходимо выполнить следующие шаги.

- Зарегистрироваться на платформе kaggle.com. Процесс выставления оценок будет проходить при подведении итогового рейтинга. Пожалуйста, укажите во вкладке Team -> Team name свои имя и фамилию в формате Имя_Фамилия (важно, чтобы имя и фамилия совпадали с данными на Stepik).
- Обучить модель, получить файл с ответами в формате .csv и сдать его в конкурс. Пробуйте и экспериментируйте. Обратите внимание, что вы можете выполнять до 20 попыток сдачи на kaggle в день.
- После окончания соревнования отправить в итоговый ноутбук с решением на степик.
- После дедлайна проверьте посылки других участников по критериям. Для этого надо зайти на степик, скачать их ноутбук и проверить скор в соревновании.

1