РЕКОМЕНДАЦИЯ ТАРИФОВ СОТОВОЙ СВЯЗИ

Учебный проект на тему "Введение в машинное обучение" курса "Специалист по Data Science" от Яндекс.Практикум.

Выполнил **Денис Абрашин** (e-mail) Версия 1.0

Оглавление

Задание

Оператор мобильной связи «Мегалайн» выяснил: многие клиенты пользуются архивными тарифами. Компания хочет построить систему, способную проанализировать поведение клиентов и предложить пользователям новый тариф: «Смарт» или «Ультра».

В вашем распоряжении данные о поведении клиентов, которые уже перешли на эти тарифы (из проекта курса «Статистический анализ данных»). Нужно построить модель для задачи классификации, которая выберет подходящий тариф. Предобработка данных не понадобится — вы её уже сделали.

Постройте модель с максимально большим значением *accuracy*. Чтобы сдать проект успешно, нужно довести долю правильных ответов по крайней мере до 0.75. Проверьте *accuracy* на тестовой выборке самостоятельно.

1. Откройте файл с данными и изучите его. Путь к файлу: `datasets/----.csv` (скачать).
2. Разделите исходные данные на обучающую, валидационную и тестовую выборки.
3. Исследуйте качество разных моделей, меняя гиперпараметры. Кратко напишите выводы **исследования**.
4. Проверьте качество модели на тестовой выборке.

5. Дополнительное задание: проверьте модели на вменяемость. Ничего страшного, если не получится: эти данные сложнее тех, с которыми вы работали раньше. В следующем курсе подробнее об этом расскажем.

Как будут проверять проект?

К чек-листу готовности проекта

Введение

В задании не зря выделено слово исследование, под которым мы будем понимать научный метод изучения чего-либо. В данной работе сосредоточим своё внимание на нескольких моделях машинного обучения и попытаемся разобраться с особенностями их применения. В качестве основного инструмента возьмем сравнение. А чтобы учебное задание выполнялось с большим интересом, устроим между моделями соревнование, результаты которого объявим в самом конце...

Название нашему соревнованию дадим на английском языке, так как участие в нем примут и иностранцы (scikit-learn):

In [1]:
```
competition_name = 'Yandex Praktikum Mini ML Contest - 2020 (Home Edition)'
```

Победителей определим, естественно, в нескольких номинациях, соответствующих популярным метрикам качества.

Подключение и настройка необходимых библиотек

In [2]:
```
# отключение предупреждений
import warnings; warnings.filterwarnings("ignore", category=Warning)
```

In [3]:
```
import os
import errno
```

In [4]:
```
from urllib.request import urlretrieve
```

In [5]:
```
from IPython.display import HTML, display
```

Настройка numpy и pandas

In [6]:
```
# обновление библиотек Numpy и Pandas для исключения ошибок версий
!pip3 install --upgrade --user --quiet --no-warn-script-location numpy==1.19.3 pandas
```

In [7]:
```
import numpy as np
import pandas as pd
```

In [8]:
```
pd.set_option('display.notebook_repr_html', True)
pd.set_option('display.max_columns', 8)
pd.set_option('display.max_rows', 10)
pd.set_option('display.width', 80)
pd.set_option('display.float_format', '{:.3f}'.format)
```

Настройка matplotlib и seaborn

In [9]:
```
# обновление библиотек
!pip3 install --upgrade --user --quiet matplotlib seaborn
```

In [10]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import seaborn as sns
```

In [11]:
```
%matplotlib inline
```

In [12]:
```
small, medium, large = 14, 16, 22
params = {'figure.figsize':   (16, 6),
          'figure.titlesize': medium,
          'legend.fontsize':  small,
          'axes.titlesize':   small,
          'axes.labelsize':   small,
          'xtick.labelsize':  small,
          'ytick.labelsize':  small,
          'legend.loc':       'best'}
plt.rcParams.update(params)
```

In [13]:
```
# повышение четкости  графиков для больших мониторов
%config InlineBackend.figure_format = 'retina'
```

In [14]:
```
# включение цветового оформления
sns.set_style('white')
```

Подключение библиотек машинного обучения

In [15]:
```
# обновление библиотек
!pip3 install --upgrade --user --quiet sklearn
```

In [16]:
```
from sklearn.model_selection import train_test_split
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree, export_text
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score, precision_score, recall_score
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV
```

In [17]:
```
!pip install --upgrade --user --quiet catboost
```

In [18]:
```
from catboost import CatBoostClassifier
```

1. Открытие и изучение файла с данными

Загрузка данных

Опишем вспомогательные функции для загрузки файлов с данными:

In [19]:
```
def load_from_url(file_url, file_path):
    """Загрузка файла из сетевого источника file_url и запись его в file_path"""
    # создаем папку для сохранения файла с данными, если она ещё не создана
    folder_path = os.path.dirname(file_path)
    if not os.path.exists(folder_path):
        print(f'Создаю папку {folder_path}')
        os.makedirs(folder_path)
```

```
    print(f'Загружаю файл из "{file_url}" в "{file_path}" ... ', end='')
    result = urlretrieve(url=file_url, filename=file_path)
    print('OK')
    return result
```

In [20]:
```
def get_data_frame(file_path, *args, **kwargs):
    """Загрузка датафрейма Pandas из локального файла file_path"""
    print(f'Открываю "{file_path}" с помощью Pandas ... ', end='')
    result = pd.read_csv(file_path, *args, **kwargs)
    print('OK')
    return result
```

In [21]:
```
# проверить наличие файла, загрузить его при отсутствии
def load_dataframe(file_url, file_path, *args, **kwargs):
    if os.path.isfile(file_path):
        print(f'Файл "{file_path}" был загружен ранее.')
    else:
        load_from_url(file_url, file_path)

    # создать датафрейм из файла
    return get_data_frame(file_path, *args, **kwargs)
```

Зададим расположение файла с данными в Интернет и локально:

In [22]:
```
dataset_url  = 'https://yandex.ru'
dataset_path = '/datasets/-----.csv'
```

Загрузим файл с данными:

In [23]:
```
raw_data = load_dataframe(dataset_url, dataset_path)
```

```
Файл "/datasets/-----.csv" был загружен ранее.
Открываю "/datasets/-----.csv" с помощью Pandas ... OK
```

Описание данных

Каждый объект в наборе данных — это информация о поведении одного пользователя за месяц.

Параметр	Описание
calls	количество звонков
minutes	суммарная длительность звонков в минутах
messages	количество sms-сообщений
mb_used	израсходованный интернет-трафик в Мб
Целевая переменная: is_ultra	каким тарифом пользовался в течение месяца («Ультра» — 1, «Смарт» — 0)

Вывод первичной информации о наборе данных

In [24]:
```
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3214 entries, 0 to 3213
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   calls     3214 non-null   float64
 1   minutes   3214 non-null   float64
 2   messages  3214 non-null   float64
 3   mb_used   3214 non-null   float64
 4   is_ultra  3214 non-null   int64  
```

```
dtypes: float64(4), int64(1)
memory usage: 125.7 KB
```

Количество полей и их названия и соответствуют описанию.

Тип данных поля `is_ultra` целый, что соответствует представлению о категориальной переменной. Осталось проверить только значения.

Тип данных полей `calls` и `messages` - число с плавающей точкой. Это не критично для планируемых к использованию алгоритмов машинного обучения. Нам необходимо лишь проверить, чтобы их дробные части равнялись нулю.

Подготовка данных для использования в машинном обучении

В задании сказано: "Предобработка данных не понадобится — вы её уже сделали". Похоже на слова будущей тёщи при первом знакомстве. Примем их к сведению, но не отступим от правила, предписывающего проводить проверку данных перед их использованием.

Из требований изучаемых моделей машинного обучения к данным выделим следующие:

- все значения должны быть числовыми (при наличии среди них категорий, их требуется некоторым способом преобразовать в числа);
- в данных не должно быть пропущенных значений (т.е. все пропущенные значения перед обучением модели следует каким-то образом заполнить или принять обоснованное решение об их удалении).

Поэтому важным этапом в предобработке любого датасета для логистической регрессии будет кодирование категориальных признаков, а так же удаление или интерпретация пропущенных значений.

Обратим особое внимание на математический смысл полученных данных (например, в них не должно быть отрицательных минут).

Все требования оформим в виде функции, которая вызывает исключение в случае невыполнения любого из требований. Если проверка прошла успешно, то функция возвращает набор данных для обучения. Так мы сможем проще организовывать пакетную обработку (pipeline):

In [25]:
```
def get_cleaned_data(raw_data):
    """Проверяет исходный набор данных и возвращает набор, пригодный для обучения"""
    # проверим все требования
    assert raw_data.select_dtypes(exclude=[np.int64, np.float64]).size == 0, 'Данные дс
    assert raw_data.min().min() >= 0,           'Все величины должны быть неотрицательн
    assert raw_data.isna().sum().sum() == 0,    'В данных не должно быть пропусков'
    assert raw_data.calls.mod(1).sum() == 0,    'Количество звонков не должно иметь дро
    assert raw_data.messages.mod(1).sum() == 0, 'Количество сообщений не должно иметь д
    assert all(raw_data.is_ultra.isin([0, 1])), 'Поле is_ultra должно принимать значени

    print('Замечаний к набору данных нет.')

    # создадим копию исходных данных
    result = raw_data.copy()

    # преобразуем типы данных к подходящему по смыслу
    result['calls'] = result['calls'].astype(np.int64)
    result['messages'] = result['messages'].astype(np.int64)

    return result
```

In [26]:
```
data = get_cleaned_data(raw_data)
```

Замечаний к набору данных нет.

In [27]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3214 entries, 0 to 3213
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   calls     3214 non-null   int64  
 1   minutes   3214 non-null   float64
 2   messages  3214 non-null   int64  
 3   mb_used   3214 non-null   float64
 4   is_ultra  3214 non-null   int64  
dtypes: float64(2), int64(3)
memory usage: 125.7 KB
```

В данных нет пропусков, другие препятствия к обучению отсутствуют.

Распределения признаков и целевой переменной

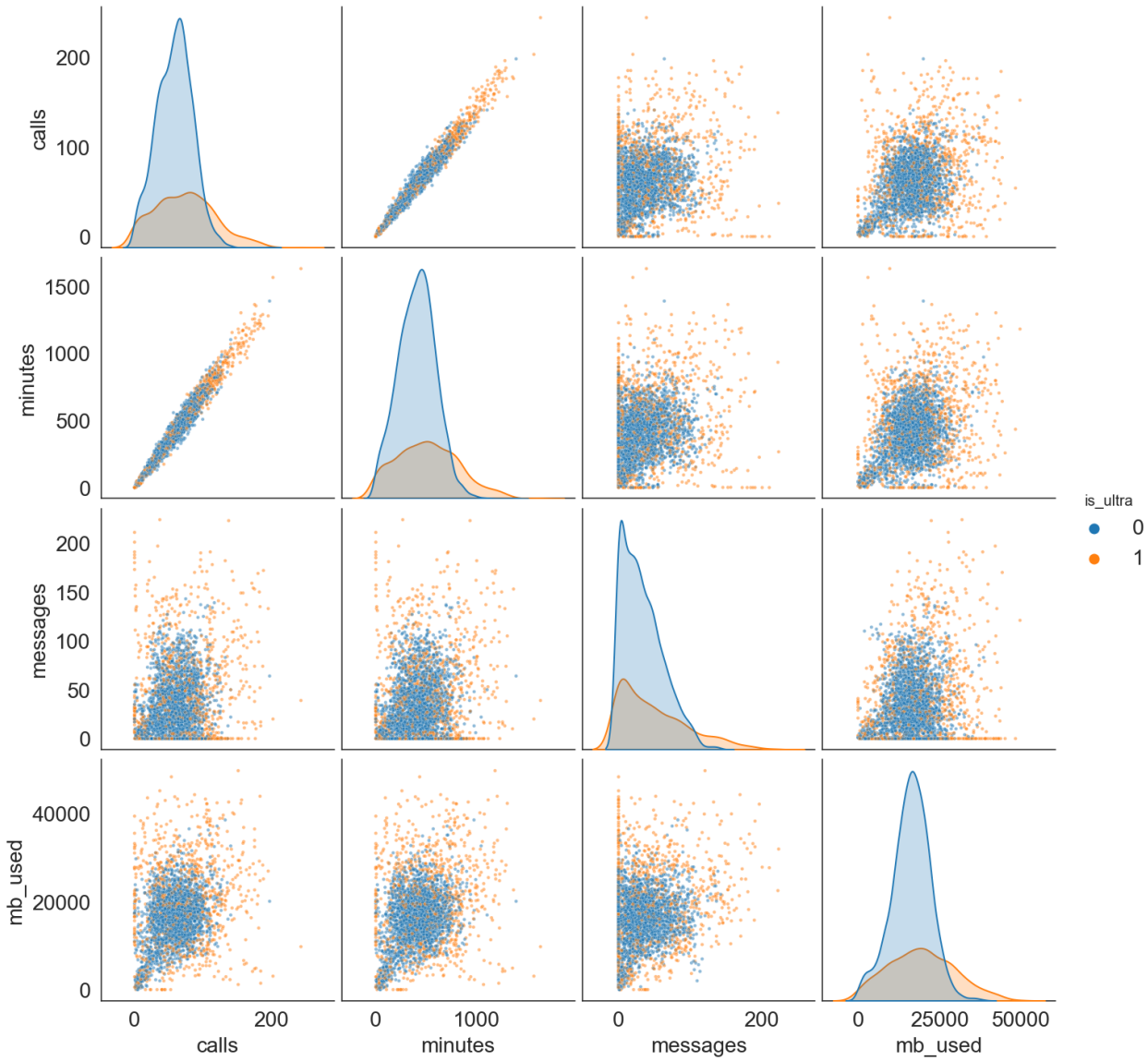
Бросим взгляд на распределения данных и корреляции:

In [28]:
```
data.describe()
```

Out[28]:

	calls	minutes	messages	mb_used	is_ultra
count	3214.000	3214.000	3214.000	3214.000	3214.000
mean	63.039	438.209	38.281	17207.674	0.306
std	33.236	234.570	36.148	7570.968	0.461
min	0.000	0.000	0.000	0.000	0.000
25%	40.000	274.575	9.000	12491.903	0.000
50%	62.000	430.600	30.000	16943.235	0.000
75%	82.000	571.927	57.000	21424.700	1.000
max	244.000	1632.060	224.000	49745.730	1.000

In [29]:
```
sns.pairplot(data, hue='is_ultra', plot_kws={'s': 5, 'alpha': 0.5});
```



In [30]:
```
data.corr()
```

Out[30]:

	calls	minutes	messages	mb_used	is_ultra
calls	1.000	0.982	0.177	0.286	0.207
minutes	0.982	1.000	0.173	0.281	0.207
messages	0.177	0.173	1.000	0.196	0.204
mb_used	0.286	0.281	0.196	1.000	0.199
is_ultra	0.207	0.207	0.204	0.199	1.000

Ну, и зачем мы вывели эти таблицы и графики? Странный вопрос, да? Даже шатающийся по курсам на платформах "Шажок" и "Бегунок" джун уже уловил несколько нюансов, которые могут оказать влияние на качество обучения:

- количество звонков (calls) и суммарная длительность звонков в минутах (minutes) имеют очень сильную корреляцию (интересно будет посмотреть, как модели учтут это);
- израсходованный интернет-трафик в Мб (mb_used) имеет больший масштаб значений по сравнению с другими переменными (как это повлияет на сходимость логистической регрессии?);
- наблюдается дисбаланс классов в целевой переменной - пользователей с тарифом «Смарт» гораздо больше абонентов «Ультра» (Очень важно учесть это при разделении данных на обучающую, валидационную и тестовую выборки. Кроме того, разные метрики

качества по-разному относятся к ошибкам в определении представителей маленьких классов);
- ни одна пара признаков не позволяет эффективно разделить выборку по значению целевой переменной (Придумать легко наивный алгоритм классификации точно не удастся, а линейные модели, скорее всего, столкнуться с определенными сложностями в решении задачи). Даже визуально не удается построить хотя бы приблизительно возможные линии, отделяющие синие точки («Смарт») от оранжевых («Ультра»).

Представляю, какой длинный список составил бы выпускник курса "Специалист по Data Science" от Яндекс.Практикум!

Вывод по шагу 1

- Полученный файл имеет корректный формат, удобный для обработки с помощью библиотеки Pandas.
- Файл успешно загружен в память.
- Названия столбцов не искажены, соответствуют полученному в задании описанию.
- В данных нет пропусков.
- Массив в целом пригоден для машинного обучения. На его результаты могут повлиять следующие особенности распределения признаков:
  - Количество звонков (calls) и суммарная длительность звонков в минутах (minutes) имеют очень сильную корреляцию.
  - Израсходованный интернет-трафик в Мб (mb_used) имеет больший масштаб значений по сравнению с другими переменными.
  - Наблюдается дисбаланс классов в целевой переменной - пользователей с тарифом «Смарт» гораздо больше абонентов «Ультра» Необходимо учесть это при разделении данных на обучающую, валидационную и тестовую выборки.

2. Разделение данных на обучающую, валидационную и тестовую выборки

Для многократной воспроизводимости полученных результатов зададим параметр генератора случайных чисел:

In [31]:
```
random_state=11122020
```

Отделение признаков от целевой переменной

In [32]:
```
def split_categories(data, target_column):
    return data.drop(columns=[target_column]), data[target_column]
```

In [33]:
```
X, y = split_categories(data, 'is_ultra')
```

Выделение тестовой выборки

Выделим 25% полученного массива данных для использования при оценке качества моделей (в финале нашего соревнования) с сохранением баланса классов в целевой переменной (y). 75% будем использовать для обучения моделей:

In [34]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.25,
                                                    random_state=random_state)
```

Убедимся, что баланс классов сохранен, сравнив численно доли абонентов «Ультра» в исходной, обучающей и тестовой выборках:

In [35]:
```
def check_ultra_part(array_name, array):
    print(f'Доля абонентов «Ультра» в {array_name} выборке: {array.sum() / len(array) *
```

In [36]:
```
check_ultra_part('исходной', y)
check_ultra_part('обучающей и валидационной', y_train)
check_ultra_part('тестовой', y_test)
```

```
Доля абонентов «Ультра» в исходной выборке: 30.6%
Доля абонентов «Ультра» в обучающей и валидационной выборке: 30.7%
Доля абонентов «Ультра» в тестовой выборке: 30.6%
```

`X_test` и `y_test` откладываем до этапа проверки моделей.

Поставщик данных против машинного обучения, или пчёлы против мёда

Часто, когда начинающий специалист хочет произвести впечатление на окружающих и убедить их в высоком уровне своих знаний, он выдает им много "дельных" советов. Например: "*Зачем тебе либы датасет сплитить? Я взял слайс айлоком в трейн 75% фичей, и столько же в один эррей с таргетом. Сцикитлёрн для слабаков!*"

Может он и прав... Нужно это проверить. Для красоты эксперимента предположим, что поставщик данных, для красоты опять же, отсортировал их по полю `calls`.

In [37]:
```
data_for_iq_test = pd.concat([X_train, y_train], axis=1).sort_values('calls')
X_train_for_iq_test, y_train_for_iq_test = split_categories(data_for_iq_test, 'is_ultra
```

Посчитаем количество образцов для тестовой выборки размером 75%:

In [38]:
```
train_count = round(y_train_for_iq_test.shape[0] * 0.75)
train_count
```

Out[38]:
```
1808
```

Создадим несбалансированные тестовую и валидационную выборки по методу "специалиста":

In [39]:
```
X_train_ub = X_train_for_iq_test.iloc[:train_count]
X_valid_ub = X_train_for_iq_test.iloc[train_count:]
y_train_ub = y_train_for_iq_test.iloc[:train_count]
y_valid_ub = y_train_for_iq_test.iloc[train_count:]
```

Убедимся в наличии дисбаланса классов:

In [40]:
```
check_ultra_part('исходной', y)
check_ultra_part('обучающей', y_train_ub)
check_ultra_part('валидационной', y_valid_ub)
```

```
Доля абонентов «Ультра» в исходной выборке: 30.6%
Доля абонентов «Ультра» в обучающей выборке: 23.9%
Доля абонентов «Ультра» в валидационной выборке: 51.0%
```

"Специалисту" в этот раз "повезло". Доли классов в обучающей и валидационной выборке оказались разными. Посмотрим, как обучится модель при таких условиях.

Балансировка обучающей и тестовой выборок

Выделим 75% обучающего массива данных собственно для обучения, а оставшиеся 25% для использования при предварительной оценке качества моделей (валидации) с сохранением

баланса классов в целевой переменной:

In [41]:
```
X_train_b, X_valid_b, y_train_b, y_valid_b = train_test_split(X_train, y_train, stratif
                                                              random_state=random_state
```

Убедимся, что баланс классов сохранен, сравнив доли абонентов «Ультра» в исходной, обучающей и валидационной выборках:

In [42]:
```
check_ultra_part('исходной', y)
check_ultra_part('обучающей', y_train_b)
check_ultra_part('валидационной', y_valid_b)
```

```
Доля абонентов «Ультра» в исходной выборке: 30.6%
Доля абонентов «Ультра» в обучающей выборке: 30.7%
Доля абонентов «Ультра» в валидационной выборке: 30.7%
```

Вывод по шагу 2

В результате подготовлено несколько выборок, которые на этапе исследования моделей позволят не только сравнить результаты различных методов машинного обучения, но и изучить влияние на них фактора баланса классов на этапе обучения и тестирования:

Выборка	Описание
X_train, y_train	Сбалансированная обучающая выборка (3/4 от исходной) - применим для кросс-валидации
X_train_ub, y_train_ub	Несбалансированная обучающая выборка (9/16 от исходной) - применим для кросс-валидации
X_valid_ub, y_valid_ub	Несбалансированная валидационная выборка (3/16 от исходной)
X_train_b, y_train_b	Сбалансированная обучающая выборка (9/16 от исходной)
X_valid_b, y_valid_b	Сбалансированная валидационная выборка (3/16 от исходной)
X_test, y_test	Сбалансированная тестовая выборка (1/4 от исходной) - применим для финальной проверки моделей

3. Исследование моделей

Вспомогательные инструменты

Описание сценариев конвейерной обработки

Библиотека `scikit-learn` предоставляет достаточно гибкие средства для организации конвейерной обработки данных при обучении и использовании моделей. Применение так называемых "трубопроводов" (`pipline`) облегчит нам исследование.

В учебных целях опишем сценарий заполнения пропусков в числовых данных медианными значениями, а также нормализации признаков (приведение значений к одинаковому масштабу для улучшения сходимости алгоритма регрессии):

In [43]:
```
numeric_features = ['calls', 'minutes', 'messages', 'mb_used']
numeric_transformer = Pipeline(
    steps=[('Заполнение пропусков', SimpleImputer(strategy='median')),
           ('Масштабирование', MinMaxScaler())])
```

In [44]:
```
preprocessor = ColumnTransformer(
```

```
    transformers=[('Обработка числовых признаков', numeric_transformer, numeric_feature
```

Подготовим лидерборд

По мере исследования различных моделей будем отбирать из них кандидатов на звание лучшей и помещать в список для финальной проверки на тестовых данных.

In [45]:
```
# список исследованных моделей (по структуре словарь, по смыслу - список :-)
models = dict()
```

Зададим начальное значение порога адекватности модели:

In [46]:
```
adequate_score = 0
```

Опишем функцию предоставления модели на конкурс:

In [47]:
```
def submit_model(title, model, score_train, score_valid, params=None, dummy=False):
    def print_score(data_name, score):
        print('{:>50s}: {:.3%}'.format(f'доля правильных ответов на {data_name} выборке

    print(f'Модель "{title}":')

    if params is not None:
        print(f'параметры: {params}')

    print_score('обучающей', score_train)
    print_score('валидационной', score_valid)

    score_test  = model.score(X_test, y_test)
    print_score('тестовой', score_test)

    y_pred = model.predict(X_test)

    models[title] = { 'название': title, 'модель': model,
                      'на обучении': score_train, 'на валидации': score_valid, 'правиль
                      'параметры': params, 'участник': not dummy,
                      'F-мера':    f1_score(y_test, y_pred),
                      'точность':  precision_score(y_test, y_pred),
                      'полнота':   recall_score(y_test, y_pred),
                      'разница между обучением и тестом': score_train - score_test
                    }

    global adequate_score

    if dummy:
        if score_test > adequate_score:
            adequate_score = score_test
            print(f'Порог адекватности модели по доле правильных ответов увеличен до {a
    else:
        if score_test <= adequate_score:
            print('Данная модель не дотягивает до уровня адекватности '
                  f'по доле правильных ответов {adequate_score:.3%}.')
```

Теперь приступим к обучению и исследованию моделей.

Базовые модели

Опишем простые базовые модели, которые в дальнейшем используем проверки других моделей на адекватность (спойлер). Воспользуемся DummyClassifier.

Классификация случайным образом

In [48]:
```
# создадим модель, обучим её и занесем в список для дальнейшей проверки на тестовых дан
clf = DummyClassifier(strategy='uniform', random_state=random_state)
clf.fit(X_train_b, y_train_b)
```

```
submit_model('Классификация случайным образом', clf,
             clf.score(X_train_b, y_train_b), clf.score(X_valid_b, y_valid_b), dummy=Tr
```

```
Модель "Классификация случайным образом":
      доля правильных ответов на обучающей выборке: 52.684%
  доля правильных ответов на валидационной выборке: 47.761%
       доля правильных ответов на тестовой выборке: 49.627%
Порог адекватности модели по доле правильных ответов увеличен до 49.627%.
```

Низкий результат продиктован тем, что данная модель не учитывает баланс классов - соотношение абонентов с разными тарифными планами.

Классификация случайным образом с учётом баланса классов

In [49]:
```
clf = DummyClassifier(strategy='stratified', random_state=random_state)
clf.fit(X_train_b, y_train_b)
submit_model('Классификация случайным образом с учётом баланса классов', clf,
             clf.score(X_train_b, y_train_b), clf.score(X_valid_b, y_valid_b), dummy=Tr
```

```
Модель "Классификация случайным образом с учётом баланса классов":
      доля правильных ответов на обучающей выборке: 57.277%
  доля правильных ответов на валидационной выборке: 58.541%
       доля правильных ответов на тестовой выборке: 60.199%
Порог адекватности модели по доле правильных ответов увеличен до 60.199%.
```

Классификация наиболее частым значением целевого признака

In [50]:
```
clf = DummyClassifier(strategy='most_frequent', random_state=random_state)
clf.fit(X_train_b, y_train_b)
submit_model('Классификация наиболее частым значением', clf,
             clf.score(X_train_b, y_train_b), clf.score(X_valid_b, y_valid_b), dummy=Tr
```

```
Модель "Классификация наиболее частым значением":
      доля правильных ответов на обучающей выборке: 69.341%
  доля правильных ответов на валидационной выборке: 69.320%
       доля правильных ответов на тестовой выборке: 69.403%
Порог адекватности модели по доле правильных ответов увеличен до 69.403%.
```

Растём над собой! Порог адекватности существенно увеличился с учетом баланса классов. Так и машинное обучение не понадобится. 😳

Классификация методом экспертной оценки

Получив это задание, автор обошёл несколько десятков офисов продаж сотовых компаний и обратился к консультантам с одним вопросом: "*Как предсказать тариф по биллингу абонента?*" 😊

Мнения экспертов разделились. Наиболее частый ответ не будем принимать во внимание, скорее всего, он является статистическим выбросом. Пойдем в другом направлении и попытаемся предсказать тариф по степени превышения показателей при сравнении с лимитами, установленными тарифом "Смарт". Они известны нам по предыдущему проекту: в абонентскую плату включено 500 минут разговора, 50 сообщений и 15 Гб интернет-трафика.

Логично предположить, что чем больше абонент превышает лимиты Смарта, тем ближе он к Ультра. Никто не хочет в общем случае платить лишние деньги. Реализуем свой собственный алгоритм, который по обучающей выборке определит оптимальные коэффициенты превышения минут разговора, количества сообщений и мегабайтов. Построим бэйзлайн-модель:

In [51]:
```
from sklearn.base import BaseEstimator, ClassifierMixin
```

In [52]:
```
class TariffExpertSmartClassifier(BaseEstimator, ClassifierMixin):
    """Класс для предсказания тарифа по степени превышения лимитов, установленных для т
    def __set_k(self, k1, k2, k3):
```

```
        """Установка значений коэффициента превышения для минут, сообщений и мегабайтов
        self.minutes_k  = k1
        self.messages_k = k2
        self.mb_used_k  = k3

    def __init__(self, minutes=500, messages=50, mb_used=15360):
        """Конструктор получает лимиты тарифа и начальные параметры обучения"""
        self.minutes    = minutes
        self.messages   = messages
        self.mb_used    = mb_used
        self.__set_k(0.0, 0.0, 0.0)

    def get_params(self, deep=True):
        """Получение параметров модели"""
        return {'minutes': self.minutes, 'messages': self.messages, 'mb_used': self.mb_
                'minutes_k': self.minutes_k, 'messages_k': self.messages_k, 'mb_used_k'

    def __do_predict(self, X, k1, k2, k3):
        """Предсказание по заданным коэффициентам"""
        return ((X['minutes']  / self.minutes  > k1) &
                (X['messages'] / self.messages > k2) &
                (X['mb_used']  / self.mb_used  > k3))

    def fit(self, X, y):
        """Обучение модели"""
        # обнулим количество правильных ответов в лучшей модели
        n = 0
        # зададим начальные коэффициенты
        self.__set_k(0.0, 0.0, 0.0)
        print('Обучение', end='')
        # пройдемся по сетке
        k_min, k_max, k_step = 0.0, 2.0, 0.05
        for k1 in np.arange(k_min, k_max, k_step):
            print('.', end='')
            for k2 in np.arange(k_min, k_max, k_step):
                for k3 in np.arange(k_min, k_max, k_step):
                    ni = (self.__do_predict(X, k1, k2, k3) == y).sum()
                    if ni > n:
                        n = ni
                        self.__set_k(k1, k2, k3)
        print()
        return self

    def predict(self, X):
        """Предсказание"""
        return self.__do_predict(X, self.minutes_k, self.messages_k, self.mb_used_k)
```

In [53]:
```
clf = TariffExpertSmartClassifier()
clf.fit(X_train_b, y_train_b)
submit_model('Классификация по лимитам тарифа "Смарт" (бэйзлайн)', clf,
             clf.score(X_train_b, y_train_b), clf.score(X_valid_b, y_valid_b),
             params=clf.get_params(), dummy=True)
```

```
Обучение.....................................
Модель "Классификация по лимитам тарифа "Смарт" (бэйзлайн)":
параметры: {'minutes': 500, 'messages': 50, 'mb_used': 15360, 'minutes_k': 0.0, 'messag
es_k': 0.0, 'mb_used_k': 1.7000000000000002}
      доля правильных ответов на обучающей выборке: 73.935%
  доля правильных ответов на валидационной выборке: 72.637%
       доля правильных ответов на тестовой выборке: 73.632%
Порог адекватности модели по доле правильных ответов увеличен до 73.632%.
```

Получили пока рекордный результат. Тем интереснее будет проверять настоящие модели машинного обучения!

Наша базовая модель строит своё предсказание исключительно по трафику. Если потребление Интернет превышает квоту тарифа "Смарт" более чем в 1.7 раза, то образец

относится к тарифу "Ультра".

Логистическая регрессия

Логистическая регрессия или логит-модель (англ. logit model) — это статистическая модель, используемая для прогнозирования вероятности возникновения некоторого события путём его сравнения с логистической кривой. Эта регрессия выдаёт ответ в виде вероятности бинарного события (1 или 0).

Логистическая регрессия с параметрами по умолчанию

Декларируется, что LogisticRegression с параметрами по умолчанию в общем случае обеспечивает достаточно высокий результат:

In [54]:
```
clf = LogisticRegression(random_state=random_state)
clf.fit(X_train_b, y_train_b)
submit_model('Логистическая регрессия (по умолчанию)', clf,
             clf.score(X_train_b, y_train_b), clf.score(X_valid_b, y_valid_b))
```

```
Модель "Логистическая регрессия (по умолчанию)":
      доля правильных ответов на обучающей выборке: 74.765%
   доля правильных ответов на валидационной выборке: 74.295%
       доля правильных ответов на тестовой выборке: 75.249%
```

Модель по умолчанию выдала проходной балл 0.75. Примечательно, что результат на тесте выше валидации. Так бывает далеко не всегда. В это раз с выбором параметра random_state на этапе получения тестовой выборки повезло.

In [55]:
```
clf = LogisticRegression(random_state=random_state)
clf.fit(X_train_ub, y_train_ub)
submit_model('Логистическая регрессия (несбалансированная выборка)', clf,
             clf.score(X_train_ub, y_train_ub), clf.score(X_valid_ub, y_valid_ub))
```

```
Модель "Логистическая регрессия (несбалансированная выборка)":
      доля правильных ответов на обучающей выборке: 79.148%
   доля правильных ответов на валидационной выборке: 49.003%
       доля правильных ответов на тестовой выборке: 70.274%
Данная модель не дотягивает до уровня адекватности по доле правильных ответов 73.632%.
```

А вот модель, обученная на выборке с неправильным учетом баланса классов (долей пользователей разных тарифов), на тесте отработала уже хуже и не проходит даже тест на адекватность.

Логистическая регрессия с L1-регуляризатором

In [56]:
```
clf = LogisticRegression(penalty='l1', solver='liblinear', random_state=random_state)
clf.fit(X_train_b, y_train_b)
submit_model('Логистическая регрессия с L1-регуляризатором', clf,
             clf.score(X_train_b, y_train_b), clf.score(X_valid_b, y_valid_b))
```

```
Модель "Логистическая регрессия с L1-регуляризатором":
      доля правильных ответов на обучающей выборке: 74.820%
   доля правильных ответов на валидационной выборке: 74.461%
       доля правильных ответов на тестовой выборке: 75.249%
```

На тестовой выборке результаты L2-регуляризации (по умолчанию) и L1-регуляризации одинаковы.

In [57]:
```
clf = LogisticRegression(penalty='l1', solver='liblinear', random_state=random_state)
clf.fit(X_train_ub, y_train_ub)
submit_model('Логистическая регрессия (L1, несбалансированная выборка)', clf,
             clf.score(X_train_ub, y_train_ub), clf.score(X_valid_ub, y_valid_ub))
```

```
Модель "Логистическая регрессия (L1, несбалансированная выборка)":
      доля правильных ответов на обучающей выборке: 78.872%
```

```
   доля правильных ответов на валидационной выборке: 49.834%
        доля правильных ответов на тестовой выборке: 71.269%
Данная модель не дотягивает до уровня адекватности по доле правильных ответов 73.632%.
```

На несбалансированной выборке логистическая регрессия с L1-регуляризатором тоже не дотянула до адекватности.

Логистическая регрессия с предварительным масштабированием признаков

In [58]:
```
clf = Pipeline(steps=
    [('Масштабирование', preprocessor),
     ('Классификация',   LogisticRegression(random_state=random_state))])
clf.fit(X_train_b, y_train_b)
submit_model('Логистическая регрессия с масштабированием признаков', clf,
             clf.score(X_train_b, y_train_b), clf.score(X_valid_b, y_valid_b))
```

```
Модель "Логистическая регрессия с масштабированием признаков":
      доля правильных ответов на обучающей выборке: 74.931%
  доля правильных ответов на валидационной выборке: 74.461%
        доля правильных ответов на тестовой выборке: 74.876%
```

Предварительное масштабирование признаков не улучшило результат. Значит логистическая регрессия неплохо справляется с признаками, имеющими разный порядок значений.

Логистическая регрессия (L2, BFGS)

Рискну предположить, что на имеющемся наборе данных использование в логистической регрессии алгоритма Бройдена-Флетчера-Гольдфарба-Шанно (BFGS) существенно не улучшит результат:

In [59]:
```
clf = LogisticRegression(penalty='l2', solver='lbfgs', random_state=random_state)
clf.fit(X_train_b, y_train_b)
submit_model('Логистическая регрессия (L2, BFGS)', clf,
             clf.score(X_train_b, y_train_b), clf.score(X_valid_b, y_valid_b))
```

```
Модель "Логистическая регрессия (L2, BFGS)":
      доля правильных ответов на обучающей выборке: 74.765%
  доля правильных ответов на валидационной выборке: 74.295%
        доля правильных ответов на тестовой выборке: 75.249%
```

Логистическая регрессия с кросс-валидацией и автоматическим подбором параметров

Попробуем найти наилучшую модель логистической регрессии с кросс-валидацией на пять разбиений, учетом баланса классов и перебором некоторых основных параметров:

In [60]:
```
parameters = {
              'solver': ['lbfgs', 'liblinear', 'sag', 'saga'],
              'intercept_scaling': [0.5, 1.0, 1.5],
              'class_weight': [None, 'balanced'],
              'C': [0.5, 1, 1.5]
             }
```

Поиск оптимальной модели с помощью GridSearchCV может занять некоторое время. Улучшится ли результат?

In [61]:
```
model = LogisticRegression(max_iter=10000, random_state=random_state)
clf = GridSearchCV(model, parameters, scoring='accuracy', verbose=1, cv=5)
clf.fit(X_train, y_train);
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 360 out of 360 | elapsed:  2.4min finished
```

In [62]:
```
submit_model(f'Логистическая регрессия с кросс-валидацией и сеткой', clf.best_estimator
```

```
            clf.score(X_train, y_train), clf.best_score_, clf.best_params_)
```

```
Модель "Логистическая регрессия с кросс-валидацией и сеткой":
параметры: {'C': 0.5, 'class_weight': None, 'intercept_scaling': 1.5, 'solver': 'liblin
ear'}
      доля правильных ответов на обучающей выборке: 74.647%
  доля правильных ответов на валидационной выборке: 74.440%
      доля правильных ответов на тестовой выборке: 75.622%
```

Удалось найти параметры модели, которые обеспечивают наилучший результат.

Метод k-ближайших соседей

Проверим, как справится с задачей метод k-ближайших соседей. Воспользуемся классом KNeighborsClassifier.

K-ближайших соседей с параметрами по умолчанию

In [63]:
```
clf = KNeighborsClassifier()
clf.fit(X_train_b, y_train_b)
submit_model('Метод k-ближайших соседей (по умолчанию)', clf,
             clf.score(X_train_b, y_train_b), clf.score(X_valid_b, y_valid_b))
```

```
Модель "Метод k-ближайших соседей (по умолчанию)":
      доля правильных ответов на обучающей выборке: 81.738%
  доля правильных ответов на валидационной выборке: 75.124%
      доля правильных ответов на тестовой выборке: 74.876%
```

K-ближайших соседей с кросс-валидацией и автоматическим подбором параметров

Попробуем найти наилучшую модель с кросс-валидацией на пять разбиений, учетом баланса классов и перебором некоторых основных параметров, ограничив алгоритм расчета расстояния манхэттенским:

In [64]:
```
parameters = {
              'n_neighbors': range(5, 15),
              'leaf_size': range(1, 5),
              'p': [1],
              'weights': ['uniform', 'distance'],
              'algorithm': ['ball_tree', 'kd_tree', 'brute']
             }
```

Поиск оптимальной модели с помощью GridSearchCV может занять некоторое время.

Улучшится ли результат?

In [65]:
```
clf = GridSearchCV(KNeighborsClassifier(), parameters, scoring='accuracy', verbose=1, c
clf.fit(X_train, y_train);
```

```
Fitting 5 folds for each of 240 candidates, totalling 1200 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1200 out of 1200 | elapsed:   26.2s finished
```

In [66]:
```
submit_model(f'K-ближайших соседей с кросс-валидацией и сеткой', clf.best_estimator_,
             clf.score(X_train, y_train), clf.best_score_, clf.best_params_)
```

```
Модель "K-ближайших соседей с кросс-валидацией и сеткой":
параметры: {'algorithm': 'ball_tree', 'leaf_size': 1, 'n_neighbors': 9, 'p': 1, 'weight
s': 'uniform'}
      доля правильных ответов на обучающей выборке: 80.332%
  доля правильных ответов на валидационной выборке: 77.054%
      доля правильных ответов на тестовой выборке: 76.493%
```

Удалось найти параметры модели k-ближайших соседей, которые обеспечивают наилучший результат, но он далёк от идеала.

Дерево решений

Применим DecisionTreeClassifier - вариант от библиотеки sklearn.

Дерево решений с параметрами по умолчанию

In [67]:
```
clf = DecisionTreeClassifier(random_state=random_state)
clf.fit(X_train_b, y_train_b)
submit_model('Дерево решений (по умолчанию)', clf,
             clf.score(X_train_b, y_train_b), clf.score(X_valid_b, y_valid_b))
```

```
Модель "Дерево решений (по умолчанию)":
      доля правильных ответов на обучающей выборке: 100.000%
  доля правильных ответов на валидационной выборке: 71.808%
       доля правильных ответов на тестовой выборке: 69.030%
Данная модель не дотягивает до уровня адекватности по доле правильных ответов 73.632%.
```

Вот это номер, первое же дерево оказалось идиотом! Ничего удивительного мы вырастили его по жадному алгоритму, который переобучился - на 100% разбирается с обучающей выборкой, но так и не понял закономерностей отнесения к тому или иному тарифу.

Дерево решений с ограничением высоты

Попытаемся побороть переобучение, ограничив высоту дерева:

In [68]:
```
clf = DecisionTreeClassifier(max_depth=5, random_state=random_state)
clf.fit(X_train_b, y_train_b)
submit_model('Дерево решений не выше 5', clf,
             clf.score(X_train_b, y_train_b), clf.score(X_valid_b, y_valid_b), {'max_de
```

```
Модель "Дерево решений не выше 5":
параметры: {'max_depth': 5}
      доля правильных ответов на обучающей выборке: 82.125%
  доля правильных ответов на валидационной выборке: 79.934%
       доля правильных ответов на тестовой выборке: 78.358%
```

Дерево решений с подбором оптимальной высоты

Попробуем сами подобрать оптимальную высоту дерева на сбалансированной выборке:

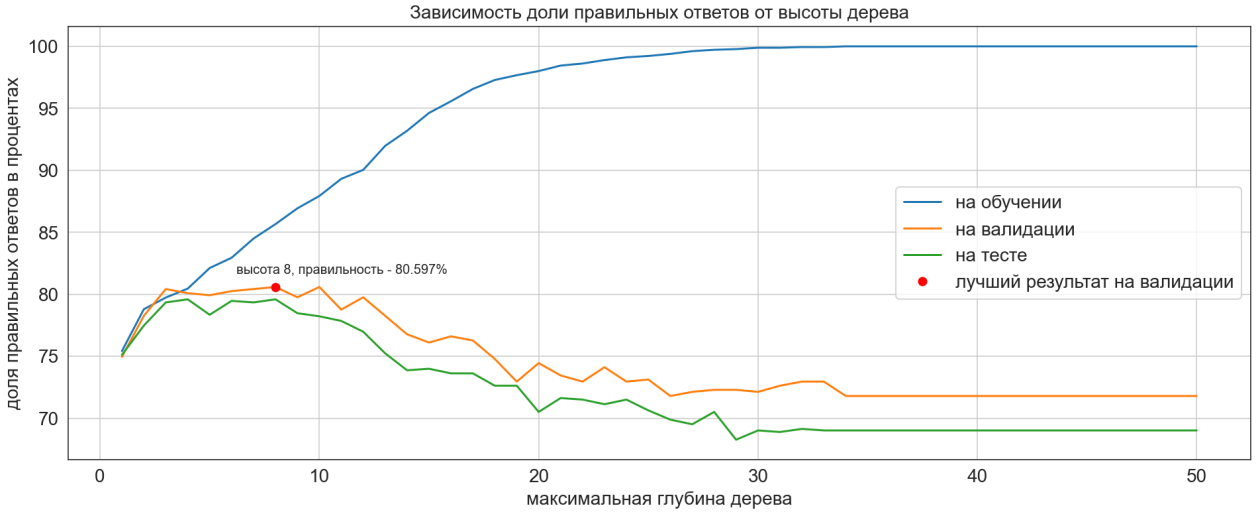
In [69]:
```
max_depth = 50
best_score = 0
best_clf = None
train_scores = []
val_scores = []
test_scores = []
for depth in range(1, max_depth+1):
    clf = DecisionTreeClassifier(max_depth=depth, random_state=random_state)
    clf.fit(X_train_b, y_train_b)
    score = clf.score(X_valid_b, y_valid_b)
    val_scores.append(score * 100)
    train_scores.append(clf.score(X_train_b, y_train_b) * 100)
    test_scores.append(clf.score(X_test, y_test) * 100)
    if score > best_score:
        best_score = score
        best_clf = clf
```

In [70]:
```
submit_model(f'Дерево решений высотой {best_clf.max_depth})', best_clf,
             best_clf.score(X_train_b, y_train_b), best_score, params={'max_depth': bes
```

```
Модель "Дерево решений высотой 8)":
параметры: {'max_depth': 8}
      доля правильных ответов на обучающей выборке: 85.667%
  доля правильных ответов на валидационной выборке: 80.597%
       доля правильных ответов на тестовой выборке: 79.602%
```

In [71]:
```
plt.plot(range(1, len(train_scores)+1), train_scores, label='на обучении')
plt.plot(range(1, len(val_scores)+1),   val_scores,   label='на валидации')
plt.plot(range(1, len(test_scores)+1),  test_scores,  label='на тесте')
```

```
y = best_score * 100
plt.annotate(f'высота {best_clf.max_depth}, правильность - {y:.3f}%', # this is the tex
                 (best_clf.max_depth, y),
                 textcoords='offset points',
                 xytext=(50,10),
                 ha='center')
plt.plot([best_clf.max_depth], [y], 'ro', label='лучший результат на валидации')
plt.xlabel('максимальная глубина дерева')
plt.ylabel('доля правильных ответов в процентах')
plt.title('Зависимость доли правильных ответов от высоты дерева')
plt.grid()
plt.legend()
plt.show()
```



Наше дерево обучалось по классической схеме для этого типа моделей: чем дерево выше, тем оно лучше замечает особенности конкретной обучающей выборки. Это приводит к переобучению - резкому снижению качества на данных, которые дерево никогда не видело. Бороться с этим можно, ограничивая высоту.

Дерево решений с кросс-валидацией и автоматическим подбором параметров

Попробуем найти наилучшую модель с кросс-валидацией на пять разбиений, учетом баланса классов и перебором некоторых основных параметров:

In [72]:
```
parameters = {
              'min_samples_split': range(2, 15),    # ограничение на количество элементо
              'max_depth': [None, 6, 7, 8, 9, 10]  # ограничение глубины
             }
```

Поиск оптимальной модели с помощью GridSearchCV может занять некоторое время. Улучшится ли результат?

In [73]:
```
model = DecisionTreeClassifier(random_state=random_state)
clf = GridSearchCV(model, parameters, scoring='accuracy', verbose=1, cv=5)
clf.fit(X_train, y_train);
```

```
Fitting 5 folds for each of 78 candidates, totalling 390 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 390 out of 390 | elapsed:    3.5s finished
```

In [74]:
```
submit_model(f'Дерево решений с кросс-валидацией и сеткой', clf.best_estimator_,
             clf.score(X_train, y_train), clf.best_score_, clf.best_params_)
```

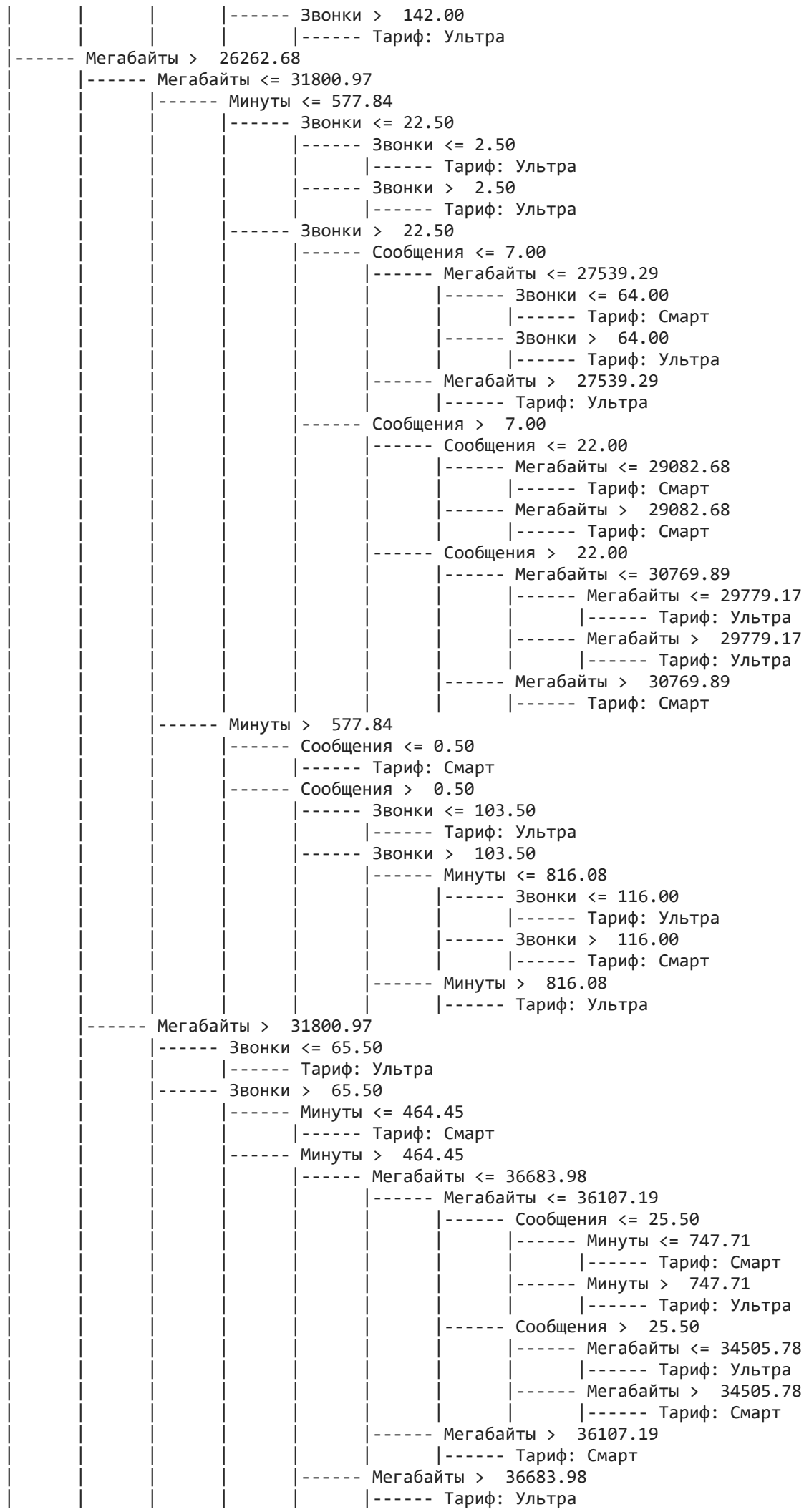
```
Модель "Дерево решений с кросс-валидацией и сеткой":
параметры: {'max_depth': 8, 'min_samples_split': 6}
      доля правильных ответов на обучающей выборке: 84.689%
  доля правильных ответов на валидационной выборке: 79.793%
      доля правильных ответов на тестовой выборке: 79.975%
```

Удалось найти параметры дерева, которые обеспечивают наилучший результат. Посмотрим на это дерево с помощью export_text, ужаснемся и похвалим растение за его логику:

In [75]:
```
text_representation = export_text(clf.best_estimator_, spacing=6,
                                  feature_names = ['Звонки', 'Минуты', 'Сообщения', 'Ме
                        .replace('class: 0', 'Тариф: Смарт') \
                        .replace('class: 1', 'Тариф: Ультра')
print(text_representation)
```

```
|------ Мегабайты <= 26262.68
|       |------ Минуты <= 738.95
|       |       |------ Сообщения <= 114.50
|       |       |       |------ Минуты <= 0.55
|       |       |       |       |------ Мегабайты <= 16889.73
|       |       |       |       |       |------ Сообщения <= 18.50
|       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |------ Сообщения >  18.50
|       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |------ Мегабайты >  16889.73
|       |       |       |       |       |------ Сообщения <= 31.00
|       |       |       |       |       |       |------ Сообщения <= 22.50
|       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |       |------ Сообщения >  22.50
|       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |------ Сообщения >  31.00
|       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |------ Минуты >  0.55
|       |       |       |       |------ Мегабайты <= 10023.95
|       |       |       |       |       |------ Минуты <= 609.86
|       |       |       |       |       |       |------ Сообщения <= 56.50
|       |       |       |       |       |       |       |------ Мегабайты <= 0.00
|       |       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |       |       |------ Мегабайты >  0.00
|       |       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |       |       |------ Сообщения >  56.50
|       |       |       |       |       |       |       |------ Сообщения <= 103.50
|       |       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |       |       |------ Сообщения >  103.50
|       |       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |       |------ Минуты >  609.86
|       |       |       |       |       |       |------ Минуты <= 693.71
|       |       |       |       |       |       |       |------ Звонки <= 105.50
|       |       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |       |       |------ Звонки >  105.50
|       |       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |       |       |------ Минуты >  693.71
|       |       |       |       |       |       |       |------ Звонки <= 101.50
|       |       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |       |       |       |------ Звонки >  101.50
|       |       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |------ Мегабайты >  10023.95
|       |       |       |       |       |------ Звонки <= 99.50
|       |       |       |       |       |       |------ Сообщения <= 7.50
|       |       |       |       |       |       |       |------ Минуты <= 534.70
|       |       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |       |       |       |------ Минуты >  534.70
|       |       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |       |       |------ Сообщения >  7.50
|       |       |       |       |       |       |       |------ Сообщения <= 73.50
|       |       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |       |       |       |------ Сообщения >  73.50
|       |       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |       |------ Звонки >  99.50
|       |       |       |       |       |       |------ Минуты <= 684.83
|       |       |       |       |       |       |       |------ Мегабайты <= 23400.01
|       |       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |       |       |------ Мегабайты >  23400.01
|       |       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |       |       |------ Минуты >  684.83
|       |       |       |       |       |       |       |------ Мегабайты <= 21008.04
```

```
|       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |       |       |------ Мегабайты >  21008.04
|       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |------ Сообщения >  114.50
|       |       |       |       |------ Минуты <= 730.84
|       |       |       |       |       |------ Сообщения <= 127.50
|       |       |       |       |       |       |------ Мегабайты <= 21921.82
|       |       |       |       |       |       |       |------ Минуты <= 364.32
|       |       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |       |       |       |------ Минуты >  364.32
|       |       |       |       |       |       |       |       |------ Звонки <= 87.00
|       |       |       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |       |       |       |------ Звонки >  87.00
|       |       |       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |       |       |------ Мегабайты >  21921.82
|       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |------ Сообщения >  127.50
|       |       |       |       |       |       |------ Звонки <= 74.50
|       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |       |------ Звонки >  74.50
|       |       |       |       |       |       |       |------ Минуты <= 549.88
|       |       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |       |       |       |------ Минуты >  549.88
|       |       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |------ Минуты >  730.84
|       |       |       |       |------ Тариф: Смарт
|       |------ Минуты >  738.95
|       |       |------ Звонки <= 129.00
|       |       |       |------ Мегабайты <= 10182.17
|       |       |       |       |------ Тариф: Ультра
|       |       |       |------ Мегабайты >  10182.17
|       |       |       |       |------ Звонки <= 109.50
|       |       |       |       |       |------ Мегабайты <= 21105.65
|       |       |       |       |       |       |------ Мегабайты <= 12703.21
|       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |       |------ Мегабайты >  12703.21
|       |       |       |       |       |       |       |------ Звонки <= 94.50
|       |       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |       |       |------ Звонки >  94.50
|       |       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |       |------ Мегабайты >  21105.65
|       |       |       |       |       |       |------ Сообщения <= 105.00
|       |       |       |       |       |       |       |------ Звонки <= 101.00
|       |       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |       |       |------ Звонки >  101.00
|       |       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |       |------ Сообщения >  105.00
|       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |------ Звонки >  109.50
|       |       |       |       |       |------ Мегабайты <= 22829.77
|       |       |       |       |       |       |------ Мегабайты <= 10818.73
|       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |       |       |------ Мегабайты >  10818.73
|       |       |       |       |       |       |       |------ Минуты <= 806.55
|       |       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |       |       |------ Минуты >  806.55
|       |       |       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |------ Мегабайты >  22829.77
|       |       |       |       |       |       |------ Звонки <= 124.50
|       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |       |       |       |       |------ Звонки >  124.50
|       |       |       |       |       |       |       |------ Тариф: Смарт
|       |       |------ Звонки >  129.00
|       |       |       |------ Звонки <= 142.00
|       |       |       |       |------ Минуты <= 1086.04
|       |       |       |       |       |------ Сообщения <= 27.00
|       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |       |------ Сообщения >  27.00
|       |       |       |       |       |       |------ Тариф: Ультра
|       |       |       |       |------ Минуты >  1086.04
|       |       |       |       |       |------ Тариф: Смарт
```


Звонки > 142.00
Тариф: Ультра
Мегабайты > 26262.68
Мегабайты <= 31800.97
Минуты <= 577.84
Звонки <= 22.50
Звонки <= 2.50
Тариф: Ультра
Звонки > 2.50
Тариф: Ультра
Звонки > 22.50
Сообщения <= 7.00
Мегабайты <= 27539.29
Звонки <= 64.00
Тариф: Смарт
Звонки > 64.00
Тариф: Ультра
Мегабайты > 27539.29
Тариф: Ультра
Сообщения > 7.00
Сообщения <= 22.00
Мегабайты <= 29082.68
Тариф: Смарт
Мегабайты > 29082.68
Тариф: Смарт
Сообщения > 22.00
Мегабайты <= 30769.89
Мегабайты <= 29779.17
Тариф: Ультра
Мегабайты > 29779.17
Тариф: Ультра
Мегабайты > 30769.89
Тариф: Смарт
Минуты > 577.84
Сообщения <= 0.50
Тариф: Смарт
Сообщения > 0.50
Звонки <= 103.50
Тариф: Ультра
Звонки > 103.50
Минуты <= 816.08
Звонки <= 116.00
Тариф: Ультра
Звонки > 116.00
Тариф: Смарт
Минуты > 816.08
Тариф: Ультра
Мегабайты > 31800.97
Звонки <= 65.50
Тариф: Ультра
Звонки > 65.50
Минуты <= 464.45
Тариф: Смарт
Минуты > 464.45
Мегабайты <= 36683.98
Мегабайты <= 36107.19
Сообщения <= 25.50
Минуты <= 747.71
Тариф: Смарт
Минуты > 747.71
Тариф: Ультра
Сообщения > 25.50
Мегабайты <= 34505.78
Тариф: Ультра
Мегабайты > 34505.78
Тариф: Смарт
Мегабайты > 36107.19
Тариф: Смарт
Мегабайты > 36683.98
Тариф: Ультра

Лучшее из одиночных деревьев, кстати, тоже начало разделение с мегабайтов, как и наша базовая модель.

Случайный лес

Попробуем использовать случайный лес RandomForestClassifier для получения еще более точных результатов.

Случайный лес с параметрами по умолчанию

In [76]:
```
clf = RandomForestClassifier(random_state=random_state)
clf.fit(X_train_b, y_train_b)
submit_model('Случайный лес (по умолчанию)', clf,
             clf.score(X_train_b, y_train_b), clf.score(X_valid_b, y_valid_b))
```

```
Модель "Случайный лес (по умолчанию)":
      доля правильных ответов на обучающей выборке: 100.000%
  доля правильных ответов на валидационной выборке: 81.924%
       доля правильных ответов на тестовой выборке: 80.721%
```

Случайный лес с ограничением высоты

In [77]:
```
clf = RandomForestClassifier(max_depth=5, random_state=random_state)
clf.fit(X_train_b, y_train_b)
submit_model('Случайный лес не выше 5', clf,
             clf.score(X_train_b, y_train_b), clf.score(X_valid_b, y_valid_b))
```

```
Модель "Случайный лес не выше 5":
      доля правильных ответов на обучающей выборке: 82.180%
  доля правильных ответов на валидационной выборке: 82.090%
       доля правильных ответов на тестовой выборке: 80.597%
```

Не стоит ограничивать деревья в лесу - качество может и снизиться вслед за ростом.

Случайный лес с кросс-валидацией и автоматическим подбором параметров

Попробуем найти наилучшую модель с кросс-валидацией на пять разбиений, учетом баланса классов и перебором некоторых основных параметров:

In [78]:
```
parameters = {
              'criterion': ['gini', 'entropy'],    # критерий Джини, энтропия
              'min_samples_split': range(5, 15)    # ограничение на количество элементо
             }
```

Поиск оптимальной модели с помощью GridSearchCV может занять некоторое время.

Улучшится ли результат?

In [79]:
```
model = RandomForestClassifier(random_state=random_state)
clf = GridSearchCV(model, parameters, scoring='accuracy', verbose=1, cv=5)
clf.fit(X_train, y_train);
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:   47.8s finished
```

In [80]:
```
submit_model(f'Случайный лес с кросс-валидацией и сеткой', clf.best_estimator_,
             clf.score(X_train, y_train), clf.best_score_, clf.best_params_)
```

```
Модель "Случайный лес с кросс-валидацией и сеткой":
параметры: {'criterion': 'gini', 'min_samples_split': 10}
      доля правильных ответов на обучающей выборке: 91.369%
  доля правильных ответов на валидационной выборке: 80.996%
       доля правильных ответов на тестовой выборке: 81.716%
```

Классификатор CatBoost

Давайте не будем проходить мимо CatBoostClassifier, который имеет аффилированность с Яндекс.

Классификатор CatBoost с параметрами по умолчанию

In [81]:
```
clf = CatBoostClassifier(random_state=random_state)
clf.fit(X_train_b, y_train_b, verbose=False, plot=False)
submit_model('CatBoost (по умолчанию)', clf,
             clf.score(X_train_b, y_train_b), clf.score(X_valid_b, y_valid_b))
```

```
Модель "CatBoost (по умолчанию)":
      доля правильных ответов на обучающей выборке: 87.659%
  доля правильных ответов на валидационной выборке: 82.753%
       доля правильных ответов на тестовой выборке: 80.846%
```

Классификатор CatBoost с особыми параметрами

Используем следующую структуру для передачи параметров обучения, которые приснились автору, когда он задремал при прохождении тренажера 😂:

In [82]:
```
params = {
    'nan_mode': 'Min', 'eval_metric': 'Logloss', 'iterations': 1000, 'sampling_frequenc
    'leaf_estimation_method': 'Newton', 'grow_policy': 'SymmetricTree', 'penalties_coef
    'boosting_type': 'Plain', 'model_shrink_mode': 'Constant', 'feature_border_type': '
    'bayesian_matrix_reg': 0.10000000149011612, 'l2_leaf_reg': 3, 'random_strength': 1,
    'rsm': 1, 'boost_from_average': False, 'model_size_reg': 0.5, 'subsample': 0.800000
    'use_best_model': False, 'class_names': [0, 1], 'depth': 6, 'posterior_sampling': F
    'border_count': 254, 'classes_count': 0, 'auto_class_weights': 'None',
    'sparse_features_conflict_fraction': 0, 'leaf_estimation_backtracking': 'AnyImprove
    'best_model_min_trees': 1, 'model_shrink_rate': 0, 'min_data_in_leaf': 1,
    'loss_function': 'Logloss', 'learning_rate': 0.01499900035560131, 'score_function':
    'task_type': 'CPU', 'leaf_estimation_iterations': 10, 'bootstrap_type': 'MVS', 'max
```

In [83]:
```
clf = CatBoostClassifier(random_state=random_state)
clf.set_params(**params)
clf.fit(X_train_b, y_train_b, verbose=False, plot=False)
submit_model('Классификатор CatBoost (особые параметры)', clf,
             clf.score(X_train_b, y_train_b), clf.score(X_valid_b, y_valid_b), params=p
```

```
Модель "Классификатор CatBoost (особые параметры)":
параметры: {'nan_mode': 'Min', 'eval_metric': 'Logloss', 'iterations': 1000, 'sampling_
frequency': 'PerTree', 'leaf_estimation_method': 'Newton', 'grow_policy': 'SymmetricTre
e', 'penalties_coefficient': 1, 'boosting_type': 'Plain', 'model_shrink_mode': 'Constan
t', 'feature_border_type': 'GreedyLogSum', 'bayesian_matrix_reg': 0.10000000149011612,
'l2_leaf_reg': 3, 'random_strength': 1, 'rsm': 1, 'boost_from_average': False, 'model_s
ize_reg': 0.5, 'subsample': 0.800000011920929, 'use_best_model': False, 'class_names':
[0, 1], 'depth': 6, 'posterior_sampling': False, 'border_count': 254, 'classes_count':
0, 'auto_class_weights': 'None', 'sparse_features_conflict_fraction': 0, 'leaf_estimati
on_backtracking': 'AnyImprovement', 'best_model_min_trees': 1, 'model_shrink_rate': 0,
'min_data_in_leaf': 1, 'loss_function': 'Logloss', 'learning_rate': 0.0149990003556013
1, 'score_function': 'Cosine', 'task_type': 'CPU', 'leaf_estimation_iterations': 10, 'b
ootstrap_type': 'MVS', 'max_leaves': 64}
      доля правильных ответов на обучающей выборке: 88.655%
  доля правильных ответов на валидационной выборке: 83.085%
       доля правильных ответов на тестовой выборке: 81.343%
```

Классификатор CatBoost с кросс-валидацией и автоматическим подбором параметров

У библиотеки CatBoost тоже есть возможность проводить кросс-валидацию и поиск оптимальных параметров по сетке. Для этого может быть использована функция grid_search. Передадим ей следующие параметры для оптимизации:

In [84]:
```
grid = {
    'learning_rate': [0.01],
    'depth': [5, 6, 7, 8],
```

```
    'l2_leaf_reg': [1, 2, 3, 4, 5, 10],
    'border_count': [200, 220, 240, 254, 260]
}
```

Попробуем найти наилучшую модель с кросс-валидацией на пять разбиений, учетом баланса классов и перебором некоторых основных параметров. Ячейки ниже не являются исполняемыми и приводятся для описания использованного затем метода:

model = CatBoostClassifier(loss_function='Logloss', random_state=random_state) model.set_params(**params) grid_search_result = model.grid_search(grid, X=X_train, y=y_train, cv=5, search_by_train_test_split=True, partition_random_seed=random_state, refit=True, shuffle=True, stratified=True, plot=False) params = model.get_all_params() print(params)Модель "Классификатор CatBoost с кросс-валидацией и поиском по сетке": параметры: {'nan_mode': 'Min', 'eval_metric': 'Logloss', 'iterations': 1000, 'sampling_frequency': 'PerTree', 'leaf_estimation_method': 'Newton', 'grow_policy': 'SymmetricTree', 'penalties_coefficient': 1, 'boosting_type': 'Plain', 'model_shrink_mode': 'Constant', 'feature_border_type': 'GreedyLogSum', 'bayesian_matrix_reg': 0.10000000149011612, 'l2_leaf_reg': 3, 'random_strength': 1, 'rsm': 1, 'boost_from_average': False, 'model_size_reg': 0.5, 'subsample': 0.800000011920929, 'use_best_model': False, 'class_names': [0, 1], 'random_seed': 11122020, 'depth': 6, 'posterior_sampling': False, 'border_count': 260, 'classes_count': 0, 'auto_class_weights': 'None', 'sparse_features_conflict_fraction': 0, 'leaf_estimation_backtracking': 'AnyImprovement', 'best_model_min_trees': 1, 'model_shrink_rate': 0, 'min_data_in_leaf': 1, 'loss_function': 'Logloss', 'learning_rate': 0.009999999776482582, 'score_function': 'Cosine', 'task_type': 'CPU', 'leaf_estimation_iterations': 10, 'bootstrap_type': 'MVS', 'max_leaves': 64} доля правильных ответов на обучающей выборке: 84.979% доля правильных ответов на валидационной выборке: 86.567% доля правильных ответов на тестовой выборке: 81.592%

Выполнение кода происходит достаточно долго (протокол приведен в приложении). Для экономии времени воспользуемся найденными оптимальными параметрами для быстрого обучения на сбалансированной выборке, но уже без кросс-валидации:

In [85]:
```
best_mesh_params = {
    'nan_mode': 'Min', 'eval_metric': 'Logloss', 'iterations': 1000, 'sampling_frequenc
    'leaf_estimation_method': 'Newton', 'grow_policy': 'SymmetricTree', 'penalties_coef
    'boosting_type': 'Plain', 'model_shrink_mode': 'Constant', 'feature_border_type': '
    'bayesian_matrix_reg': 0.10000000149011612, 'l2_leaf_reg': 3, 'random_strength': 1,
    'boost_from_average': False, 'model_size_reg': 0.5, 'subsample': 0.800000011920929,
    'use_best_model': False, 'class_names': [0, 1], 'random_seed': 11122020, 'depth': 6
    'posterior_sampling': False, 'border_count': 260, 'classes_count': 0, 'auto_class_w
    'sparse_features_conflict_fraction': 0, 'leaf_estimation_backtracking': 'AnyImprove
    'best_model_min_trees': 1, 'model_shrink_rate': 0, 'min_data_in_leaf': 1,
    'loss_function': 'Logloss', 'learning_rate': 0.009999999776482582, 'score_function'
    'task_type': 'CPU', 'leaf_estimation_iterations': 10, 'bootstrap_type': 'MVS', 'max
```

In [86]:
```
clf = CatBoostClassifier()
clf.set_params(**best_mesh_params)
clf.fit(X_train_b, y_train_b, verbose=False, plot=False)
submit_model('CatBoost с параметрами от кросс-валидации', clf,
             clf.score(X_train_b, y_train_b), clf.score(X_valid_b, y_valid_b), params=p
```

```
Модель "CatBoost с параметрами от кросс-валидации":
параметры: {'nan_mode': 'Min', 'eval_metric': 'Logloss', 'iterations': 1000, 'sampling_
frequency': 'PerTree', 'leaf_estimation_method': 'Newton', 'grow_policy': 'SymmetricTre
e', 'penalties_coefficient': 1, 'boosting_type': 'Plain', 'model_shrink_mode': 'Constan
t', 'feature_border_type': 'GreedyLogSum', 'bayesian_matrix_reg': 0.10000000149011612,
'l2_leaf_reg': 3, 'random_strength': 1, 'rsm': 1, 'boost_from_average': False, 'model_s
ize_reg': 0.5, 'subsample': 0.800000011920929, 'use_best_model': False, 'class_names':
[0, 1], 'depth': 6, 'posterior_sampling': False, 'border_count': 254, 'classes_count':
0, 'auto_class_weights': 'None', 'sparse_features_conflict_fraction': 0, 'leaf_estimati
on_backtracking': 'AnyImprovement', 'best_model_min_trees': 1, 'model_shrink_rate': 0,
'min_data_in_leaf': 1, 'loss_function': 'Logloss', 'learning_rate': 0.0149990003556013
1, 'score_function': 'Cosine', 'task_type': 'CPU', 'leaf_estimation_iterations': 10, 'b
ootstrap_type': 'MVS', 'max_leaves': 64}
      доля правильных ответов на обучающей выборке: 85.778%
  доля правильных ответов на валидационной выборке: 82.255%
       доля правильных ответов на тестовой выборке: 81.468%
```

Вывод по шагу 3

Любая модель машинного обучения имеет свои параметры, от правильности подбора которых зависит длительность и качество обучения.

Качество зависит также от особенностей внутренней реализация методов машинного обучения. В нашем случае линейные модели хуже справились с массивом данных, в котором слабая корреляция между признаками и целевой переменной. В этой ситуации иногда может помочь введение синтетических признаков (переход в новое признаковое пространство).

4. Проверка моделей на тестовой выборке

Подсчет результатов

Посчитаем результаты, показанные различными моделями машинного обучения:

In [87]:
```
score_table = pd.DataFrame.from_dict(data=models, orient='index').sort_values('правильн
```

Теперь в нашем распоряжении есть вся необходимая информация об обучении моделей, собранная в набор данных Pandas:

In [88]:
```
score_table.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 23 entries, Классификация случайным образом to Случайный лес с кросс-валидацией
и сеткой
Data columns (total 11 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   название                          23 non-null     object
 1   модель                            23 non-null     object
 2   на обучении                       23 non-null     float64
 3   на валидации                      23 non-null     float64
 4   правильность                      23 non-null     float64
 5   параметры                         9 non-null      object
 6   участник                          23 non-null     bool
 7   F-мера                            23 non-null     float64
 8   точность                          23 non-null     float64
 9   полнота                           23 non-null     float64
 10  разница между обучением и тестом  23 non-null     float64
dtypes: bool(1), float64(7), object(3)
memory usage: 2.0+ KB
```

Опишем функцию вывода результатов:

In [89]:
```
def show_winner(title, field, is_max=True):
    """Вывод победителя в номинации"""
    if is_max:
        winner = score_table[field].idxmax()
    else:
        winner = score_table[field].idxmin()
    score  = score_table.loc[winner][field]
    display(HTML(f'<b>{title} - {winner}</b> ({field} - {score:.3f})'))

def show_model_table(models_table):
    """Вывод таблицы моделей"""
    with pd.option_context('display.max_rows', models_table.shape[0]):
        display(models_table[['правильность', 'F-мера', 'на валидации', 'на обучении',

def show_contest_results():
    """Вывод всех результатов"""
    display(HTML(f'<br><h3>Результаты соревнования "{competition_name}"</h3>'))

    show_winner('Абсолютный победитель по доле правильных ответов', 'правильность')
    show_winner('Победитель по F-мере', 'F-мера')
```

```
    show_model_table(score_table.sort_values('правильность', ascending=False))

    barwidth = 1
    ax = score_table[['на обучении']].plot.barh(color='powderblue', width=barwidth, fig
    score_table[['на валидации']].plot.barh(color='deepskyblue', width=barwidth, ax=ax)
    score_table[['правильность']].plot.barh(color='cadetblue', width=barwidth, ax=ax)
    score_table[['F-мера']].plot.barh(color='#5090a6', width=barwidth, ax=ax)

    plt.xlabel('метрика качества')
    plt.legend(loc='lower right')
    plt.grid()
    plt.title('Гистограмма оценок качества моделей по основным критериям')
    plt.show()

    display(HTML(f'<br><h3>Шнобелевские номенации соревнования "{competition_name}"</h3
    show_winner('"Любитель сессии"', 'на валидации')
    show_winner('"Мистер точность"', 'точность')
    show_winner('"Мадам полнота"', 'полнота')
    show_winner('"Зубрилка года"', 'на обучении')
    show_winner('"Завышенные ожидания"', 'разница между обучением и тестом')
    show_winner('"Мне плевать на учёбу!"', 'на обучении', is_max=False)
    show_winner('"Мне плевать на сессию!"', 'на валидации', is_max=False)
    show_winner('"Мне чихать на всё!"', 'правильность', is_max=False)
    show_winner('"Не пойду работать после учёбы!"', 'F-мера', is_max=False)
```

In [90]:
```
show_contest_results()
```

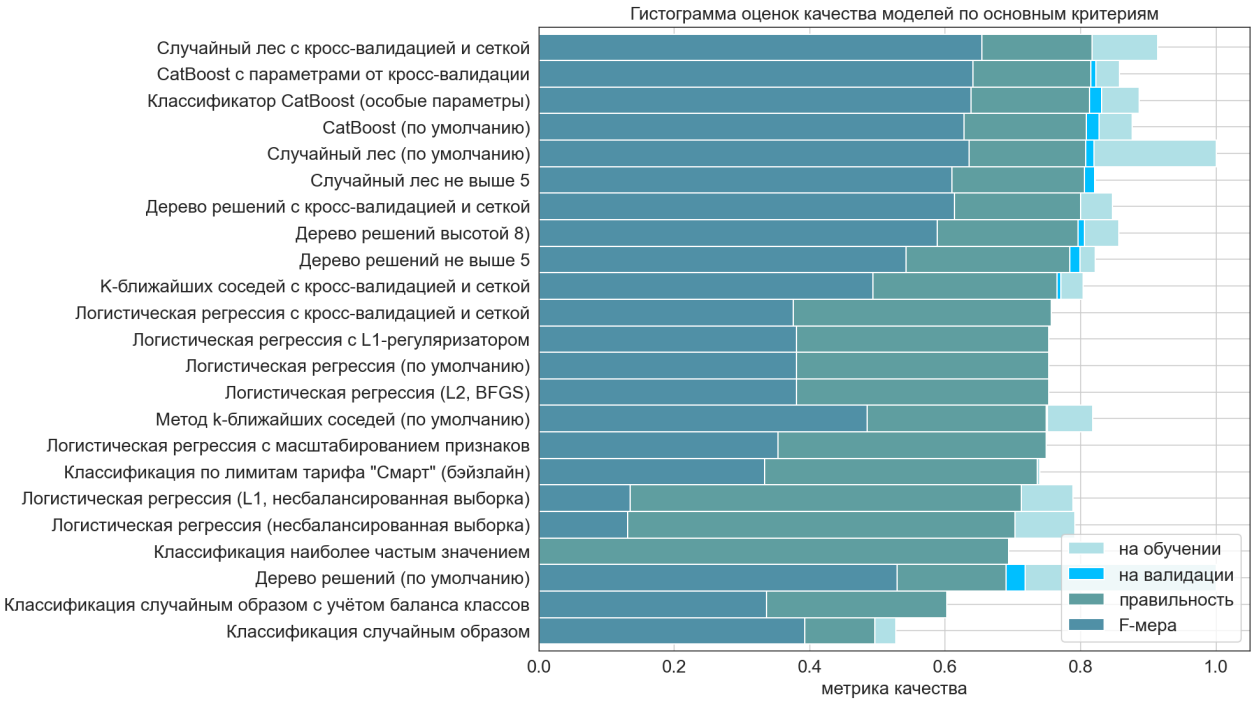
Результаты соревнования "Yandex Praktikum Mini ML Contest - 2020 (Home Edition)"

Абсолютный победитель по доле правильных ответов - Случайный лес с кросс-валидацией и сеткой (правильность - 0.817)

Победитель по F-мере - Случайный лес с кросс-валидацией и сеткой (F-мера - 0.654)

	правильность	F-мера	на валидации	на обучении	точность	полнота
Случайный лес с кросс-валидацией и сеткой	0.817	0.654	0.810	0.914	0.777	0.565
CatBoost с параметрами от кросс-валидации	0.815	0.641	0.823	0.858	0.787	0.541
Классификатор CatBoost (особые параметры)	0.813	0.638	0.831	0.887	0.786	0.537
CatBoost (по умолчанию)	0.808	0.628	0.828	0.877	0.774	0.528
Случайный лес (по умолчанию)	0.807	0.635	0.819	1.000	0.754	0.549
Случайный лес не выше 5	0.806	0.610	0.821	0.822	0.792	0.496
Дерево решений с кросс-валидацией и сеткой	0.800	0.614	0.798	0.847	0.749	0.520
Дерево решений высотой 8)	0.796	0.588	0.806	0.857	0.770	0.476
Дерево решений не выше 5	0.784	0.542	0.799	0.821	0.769	0.419
К-ближайших соседей с кросс-валидацией и сеткой	0.765	0.493	0.771	0.803	0.724	0.374
Логистическая регрессия с кросс-валидацией и сеткой	0.756	0.376	0.744	0.746	0.868	0.240

	правильность	F-мера	на валидации	на обучении	точность	полнота
Логистическая регрессия с L1-регуляризатором	0.752	0.380	0.745	0.748	0.813	0.248
Логистическая регрессия (по умолчанию)	0.752	0.380	0.743	0.748	0.813	0.248
Логистическая регрессия (L2, BFGS)	0.752	0.380	0.743	0.748	0.813	0.248
Метод k-ближайших соседей (по умолчанию)	0.749	0.485	0.751	0.817	0.651	0.386
Логистическая регрессия с масштабированием признаков	0.749	0.353	0.745	0.749	0.833	0.224
Классификация по лимитам тарифа "Смарт" (бэйзлайн)	0.736	0.333	0.726	0.739	0.736	0.215
Логистическая регрессия (L1, несбалансированная выборка)	0.713	0.135	0.498	0.789	0.857	0.073
Логистическая регрессия (несбалансированная выборка)	0.703	0.131	0.490	0.791	0.621	0.073
Классификация наиболее частым значением	0.694	0.000	0.693	0.693	0.000	0.000
Дерево решений (по умолчанию)	0.690	0.529	0.718	1.000	0.495	0.569
Классификация случайным образом с учётом баланса классов	0.602	0.336	0.585	0.573	0.343	0.329
Классификация случайным образом	0.496	0.393	0.478	0.527	0.311	0.533



Шнобелевские номенации соревнования "Yandex Praktikum Mini ML Contest - 2020 (Home Edition)"

"Любитель сессии" - Классификатор CatBoost (особые параметры) (на валидации - 0.831)

"Мистер точность" - Логистическая регрессия с кросс-валидацией и сеткой (точность - 0.868)
"Мадам полнота" - Дерево решений (по умолчанию) (полнота - 0.569)
"Зубрилка года" - Дерево решений (по умолчанию) (на обучении - 1.000)
"Завышенные ожидания" - Дерево решений (по умолчанию) (разница между обучением и тестом - 0.310)
"Мне плевать на учёбу!" - Классификация случайным образом (на обучении - 0.527)
"Мне плевать на сессию!" - Классификация случайным образом (на валидации - 0.478)
"Мне чихать на всё!" - Классификация случайным образом (правильность - 0.496)
"Не пойду работать после учёбы!" - Классификация наиболее частым значением (F-мера - 0.000)

Вывод по шагу 4

Наилучшие результаты показали модели на основе градиентного бустинга над решающими деревьями с поиском оптимальных параметров по сетке и кросс-валидации:

- Scikit-learn RandomForestClassifier;
- CatBoostClassifier.

Одиночное дерево решений без ограничения его высоты работает хуже, чем логистическая регрессия и метод k-ближайших соседей (основная причина - переобучение).

5. Проверка моделей на адекватность

Выведем список моделей, которые не смогли преодолеть порог адекватности, который мы определили при создании базовых моделей:

In [91]:
```
display(HTML(f'<br><h4>Модели, не прошедшие тест на адекватность в соревновании "{compe
stupid_models = score_table[score_table['участник'] & (score_table['правильность'] <= a
show_model_table(stupid_models.sort_values('правильность'))
display(HTML(f'Порог адекватности {adequate_score:.3f} получен на основании лучшего рез
base_models = score_table[score_table['участник'] == False]
show_model_table(base_models.sort_values('правильность', ascending=False))
```

Модели, не прошедшие тест на адекватность в соревновании "Yandex Praktikum Mini ML Contest - 2020 (Home Edition)"

	правильность	F-мера	на валидации	на обучении	точность	полнота
Дерево решений (по умолчанию)	0.690	0.529	0.718	1.000	0.495	0.569
Логистическая регрессия (несбалансированная выборка)	0.703	0.131	0.490	0.791	0.621	0.073
Логистическая регрессия (L1, несбалансированная выборка)	0.713	0.135	0.498	0.789	0.857	0.073

Порог адекватности 0.736 получен на основании лучшего результата следующих моделей:

	правильность	F-мера	на валидации	на обучении	точность	полнота
Классификация по лимитам тарифа "Смарт" (бэйзлайн)	0.736	0.333	0.726	0.739	0.736	0.215

	правильность	F-мера	на валидации	на обучении	точность	полнота
Классификация наиболее частым значением	0.694	0.000	0.693	0.693	0.000	0.000
Классификация случайным образом с учётом баланса классов	0.602	0.336	0.585	0.573	0.343	0.329
Классификация случайным образом	0.496	0.393	0.478	0.527	0.311	0.533

Вывод по шагу 5

Не любая модель может справится с поставленной задачей.

1. В нашем случае самым неадекватным оказалось переобученное одиночное дерево решений, выращенное у забора в виде обучающей выборки. Когда забор убрали, дерево под случайными порывами ветра от валидационной и тестовой выборок упало в дождевую канаву.
2. Логистическая регрессия по своей сути с трудом улавливает нелинейные закономерности. А тут ещё "специалист" обучил её на несбалансированной выборке. Результат на валидации и тесте предсказуемо оказался неадекватным.

6. Общий вывод

Построено несколько моделей машинного обучения, доля правильных ответов которых превышает установленный в задании порог 0.75.

Наилучшие результаты показали модели на основе градиентного бустинга над решающими деревьями с поиском оптимальных параметров по сетке и кросс-валидации:

- Scikit-learn RandomForestClassifier;
- CatBoostClassifier.

Одиночное дерево решений без ограничения его высоты работает хуже, чем логистическая регрессия и метод k-ближайших соседей (основная причина - переобучение).

Не любая модель может справится с поставленной задачей. В нашем случае самыми неадекватными оказались склонное к переобучению одиночное дерево решений и логистическая регрессия по своей сути с трудом улавливающая нелинейные закономерности.

Чек-лист готовности проекта

- [x] Jupyter Notebook открыт
- [x] Весь код исполняется без ошибок
- [x] Ячейки с кодом расположены в порядке исполнения
- [x] Выполнено задание 1: данные загружены и изучены
- [x] Выполнено задание 2: данные разбиты на три выборки
- [x] Выполнено задание 3: проведено исследование моделей
  - [x] Рассмотрено больше одной модели (1, 2, 3, 4, 5, 6)
  - [x] Рассмотрено хотя бы 3 значения гипепараметров для какой-нибудь модели
  - [x] Написаны выводы по результатам исследования

- [x] Выполнено задание 4: Проведено тестирование
- [x] Удалось достичь accuracy не меньше 0.75

К заданию

Как будут проверять проект?

Мы подготовили критерии оценки проекта, которыми руководствуются ревьюеры. Прежде чем приступить к решению кейса, внимательно их изучите. На что обращают внимание ревьюеры, проверяя проект:

- Как вы изучаете данные после загрузки?
- Корректно ли разделяете данные на выборки?
- Как выбираете размеры выборок?
- Правильно ли вы оцениваете качество моделей в исследовании?
- Какие модели и гиперпараметры вы используете?
- Какие выводы об исследовании делаете?
- Правильно ли тестируете модели?
- Насколько высокое значение accuracy получаете?
- Соблюдаете структуру проекта и поддерживаете аккуратность кода?

К заданию

Таблица контроля версий

Версия	Дата	Описание
1.0	12.12.2020	Направлено на первичную проверку

К оглавлению

Спасибо за проверку!

Приложение

Протокол подбора параметров CatBoost по сетке и кросс-валидации

Выполнение кода происходит достаточно долго. Здесь приведем только протокол работы.

Общие параметры зададим через словарь:

```
params = {
    'nan_mode': 'Min',
    'eval_metric': 'Logloss',
    'iterations': 1000,
    'sampling_frequency': 'PerTree',
    'leaf_estimation_method': 'Newton',
    'grow_policy': 'SymmetricTree',
    'penalties_coefficient': 1,
    'boosting_type': 'Plain',
    'model_shrink_mode': 'Constant',
    'feature_border_type': 'GreedyLogSum',
    'bayesian_matrix_reg': 0.10000000149011612,
```

```
    'l2_leaf_reg': 3,
    'random_strength': 1,
    'rsm': 1,
    'boost_from_average': False,
    'model_size_reg': 0.5,
    'subsample': 0.800000011920929,
    'use_best_model': False,
    'class_names': [0, 1],
    'depth': 6,
    'posterior_sampling': False,
    'border_count': 254,
    'classes_count': 0,
    'auto_class_weights': 'None',
    'sparse_features_conflict_fraction': 0,
    'leaf_estimation_backtracking': 'AnyImprovement',
    'best_model_min_trees': 1,
    'model_shrink_rate': 0,
    'min_data_in_leaf': 1,
    'loss_function': 'Logloss',
    'learning_rate': 0.01499900035560131,
    'score_function': 'Cosine',
    'task_type': 'CPU',
    'leaf_estimation_iterations': 10,
    'bootstrap_type': 'MVS',
    'max_leaves': 64
}
```

Обучим модель с начальными параметрами на сбалансированной выборке:

```
clf = CatBoostClassifier(random_state=random_state)
clf.set_params(**params)
clf.fit(X_train_b, y_train_b, verbose=False, plot=False)
```

Модель CatBoost с начальными параметрами показала следующий результат:

- доля правильных ответов на обучающей выборке - 88.655%
- доля правильных ответов на валидационной выборке - 83.085%
- **доля правильных ответов на тестовой выборке - 81.343%**

У библиотеки CatBoost тоже есть возможность проводить кросс-валидацию и поиск оптимальных параметров по сетке. Для этого может быть использована функция grid_search. Передадим ей следующие параметры для оптимизации:

```
grid = {
    'learning_rate': [0.01],
    'depth': [5, 6, 7, 8],
    'l2_leaf_reg': [1, 2, 3, 4, 5, 10],
    'border_count': [200, 220, 240, 254, 260]
}
```

Попробуем найти наилучшую модель с кросс-валидацией на пять разбиений, учетом баланса классов и перебором некоторых основных параметров.

```
model = CatBoostClassifier(loss_function='Logloss', random_state=random_state) model.set_params(**params)
grid_search_result = model.grid_search(grid, X=X_train, y=y_train, cv=5, search_by_train_test_split=True,
partition_random_seed=random_state, refit=True, shuffle=True, stratified=True, plot=False)
```

Протокол работы:

```
bestTest = 0.4309875974 bestIteration = 956 0: loss: 0.4309876 best: 0.4309876 (0) total: 2.08s remaining: 4m 7s
bestTest = 0.4292342234 bestIteration = 999 1: loss: 0.4292342 best: 0.4292342 (1) total: 4.13s remaining: 4m 3s
bestTest = 0.4324581679 bestIteration = 979 2: loss: 0.4324582 best: 0.4292342 (1) total: 6.25s remaining: 4m 3s
bestTest = 0.4317836832 bestIteration = 999 3: loss: 0.4317837 best: 0.4292342 (1) total: 8.37s remaining: 4m 2s
bestTest = 0.4322972087 bestIteration = 998 4: loss: 0.4322972 best: 0.4292342 (1) total: 10.4s remaining: 3m 59s
bestTest = 0.4318253488 bestIteration = 973 5: loss: 0.4318253 best: 0.4292342 (1) total: 12.6s remaining: 4m
bestTest = 0.4321965438 bestIteration = 998 6: loss: 0.4321965 best: 0.4292342 (1) total: 14.9s remaining: 4m
bestTest = 0.4325996791 bestIteration = 990 7: loss: 0.4325997 best: 0.4292342 (1) total: 17.2s remaining: 4m
bestTest = 0.4337857657 bestIteration = 999 8: loss: 0.4337858 best: 0.4292342 (1) total: 19.5s remaining: 4m 1s
bestTest = 0.4326233355 bestIteration = 974 9: loss: 0.4326233 best: 0.4292342 (1) total: 21.7s remaining: 3m 58s
bestTest = 0.4327608364 bestIteration = 999 10: loss: 0.4327608 best: 0.4292342 (1) total: 24s remaining: 3m 57s
bestTest = 0.4343786122 bestIteration = 983 11: loss: 0.4343786 best: 0.4292342 (1) total: 26.1s remaining: 3m 55s
bestTest = 0.4336418974 bestIteration = 887 12: loss: 0.4336419 best: 0.4292342 (1) total: 28.6s remaining: 3m 55s
bestTest = 0.4324577301 bestIteration = 954 13: loss: 0.4324577 best: 0.4292342 (1) total: 30.9s remaining: 3m 54s
bestTest = 0.4326985028 bestIteration = 891 14: loss: 0.4326985 best: 0.4292342 (1) total: 33.2s remaining: 3m 52s
bestTest = 0.4335865841 bestIteration = 917 15: loss: 0.4335866 best: 0.4292342 (1) total: 35.5s remaining: 3m 51s
bestTest = 0.4328498976 bestIteration = 999 16: loss: 0.4328499 best: 0.4292342 (1) total: 37.9s remaining: 3m 49s
bestTest = 0.4331201049 bestIteration = 998 17: loss: 0.4331201 best: 0.4292342 (1) total: 40.2s remaining: 3m 47s
bestTest = 0.4300281787 bestIteration = 994 18: loss: 0.4300282 best: 0.4292342 (1) total: 42.9s remaining: 3m 48s
bestTest = 0.4328650657 bestIteration = 860 19: loss: 0.4328651 best: 0.4292342 (1) total: 45.4s remaining: 3m 46s
bestTest = 0.4323883533 bestIteration = 992 20: loss: 0.4323884 best: 0.4292342 (1) total: 47.7s remaining: 3m 44s
bestTest = 0.4323673861 bestIteration = 997 21: loss: 0.4323674 best: 0.4292342 (1) total: 50s remaining: 3m 42s
bestTest = 0.4315020847 bestIteration = 993 22: loss: 0.4315021 best: 0.4292342 (1) total: 52.2s remaining: 3m 40s
bestTest = 0.4329837927 bestIteration = 999 23: loss: 0.4329838 best: 0.4292342 (1) total: 54.7s remaining: 3m 38s
bestTest = 0.4293917322 bestIteration = 991 24: loss: 0.4293917 best: 0.4292342 (1) total: 57.5s remaining: 3m 38s
bestTest = 0.4302178047 bestIteration = 967 25: loss: 0.4302178 best: 0.4292342 (1) total: 59.8s remaining: 3m 36s
bestTest = 0.4280054367 bestIteration = 893 26: loss: 0.4280054 best: 0.4280054 (26) total: 1m 2s remaining: 3m 33s
bestTest = 0.4308377115 bestIteration = 923 27: loss: 0.4308377 best: 0.4280054 (26) total: 1m 4s remaining: 3m 31s
bestTest = 0.4304663628 bestIteration = 976 28: loss: 0.4304664 best: 0.4280054 (26) total: 1m 6s remaining: 3m 29s
bestTest = 0.4309515692 bestIteration = 980 29: loss: 0.4309516 best: 0.4280054 (26) total: 1m 9s remaining: 3m 27s
Estimating final quality...
```

Наилучшие результаты получены для следующих параметров:

```
params = model.get_all_params() print(params) {'nan_mode': 'Min', 'eval_metric': 'Logloss', 'iterations': 1000,
'sampling_frequency': 'PerTree', 'leaf_estimation_method': 'Newton', 'grow_policy': 'SymmetricTree',
'penalties_coefficient': 1, 'boosting_type': 'Plain', 'model_shrink_mode': 'Constant', 'feature_border_type':
'GreedyLogSum', 'bayesian_matrix_reg': 0.10000000149011612, 'l2_leaf_reg': 3, 'random_strength': 1, 'rsm': 1,
'boost_from_average': False, 'model_size_reg': 0.5, 'subsample': 0.800000011920929, 'use_best_model': False,
'class_names': [0, 1], 'random_seed': 11122020, 'depth': 6, 'posterior_sampling': False, 'border_count': 260,
'classes_count': 0, 'auto_class_weights': 'None', 'sparse_features_conflict_fraction': 0, 'leaf_estimation_backtracking':
'AnyImprovement', 'best_model_min_trees': 1, 'model_shrink_rate': 0, 'min_data_in_leaf': 1, 'loss_function': 'Logloss',
'learning_rate': 0.009999999776482582, 'score_function': 'Cosine', 'task_type': 'CPU', 'leaf_estimation_iterations': 10,
'bootstrap_type': 'MVS', 'max_leaves': 64}
```

Результаты модели классификатора CatBoost с кросс-валидацией и поиском по сетке:

- доля правильных ответов на обучающей выборке: 84.979%
- доля правильных ответов на валидационной выборке: 86.567%
- **доля правильных ответов на тестовой выборке: 81.592%**

Если бы в нашем соревновании не было ограничения по времени, то этот результат был бы лучшим для CatBoost 😎. Используем эти параметры для быстрой подготовки другой модели к соревнованию (параметры используем от кросс-валидации, а обучимся быстро только на сбалансированной выборке). Назад в будущее!

К оглавлению