# Отчёт

по проекту

# «Идентификация пользователей»

В рамках 6 курса специализции Coursera

«Машинное обучение и анализ данных»

Автор Субботин Сергей Александрович

Сентябрь, 2020

# Содержание

**1 Описание проекта** **2**

**2 Предобработка данных** **4**

**3 Анализ признаков** **8**

**4 Пробные обучения основных алгоритмов** **15**

**5 Выбор основной модели и обучение классификатора** **16**
5.1 Создание признаков . . . . . 16
5.2 Схема валидации моделей . . . . . 18
5.3 Выбор алгоритма классификации . . . . . 19
5.4 Выбор схемы векторизации . . . . . 21
5.5 Настройка параметра регуляризации . . . . . 22
5.6 Подбор дополнительных признаков . . . . . 23
5.7 Обучение итоговой модели . . . . . 25

**6 Заключение** **28**

# 1 Описание проекта

Рассматривается широкий класс задач идентификации «пользователей» по их поведению.

Например, компания Яндекс решает задачу идентификации взломщика почтового ящика по его поведению. Взломщик будет себя вести не так, как владелец ящика: он может не удалять сообщения сразу по прочтении, он будет по-другому ставить флажки сообщениям и даже по-своему двигать мышкой. Тогда такого злоумышленника можно идентифицировать и «выкинуть» из почтового ящика, предложив хозяину войти по SMS-коду.

Другой пример из той же области — системы обнаружения/предотвращения вторжений (IPS/IDS), где происходит детектирование нестандартного поведения пользователя/хоста в защищаемой сети, и в случае выявления аномалий приниматься заданные действия по уведомлению администраторов или изоляции подозрительного хоста в сети.

В данном проекте будет решаться задача классификации пользователя по истории посещений сайтов. Целью проекта является продемонстрировать навыки работы с данными, их предобработкой, анализом и извлечением дополнительных признаков для этих данных; применение алгоритмов машинного обучения, построения моделей и их валидации и др.

В проекте используются данные, полученные из мастер-датасета cez13-0.data. Этот датасет сформрован из истории запросов полученных на прокси-сервере университета Блеза Паскаля (Клермон-Ферране, Франция), за период с ноября 2013 по май 2014 года и содержит записи посещений сайтов для 3388 пользователей. Каждый объект мастер-датасета содержит следующие признаки:

- id пользователя
- метку времени (дата и время) посещения сайта
- URL или IP-адрес запрошенной страницы.

Запросы в мастер-датасете отфильтрованы, исключены при этом запросы из адресных списков http://winhelp2002.mvps.org/hosts.htm и https://pgl.yoyo.org/as (реклама, баннеры, счётчики и т.п.), а так же сайты содержащие слова из вручную сформированного списка ключевых слов (напр. «ads», «suggestqueries.google.com», «cdn.» и т.п).

На основе данных мастер-датасета составителями задания для проекта сформированы следующие выборки:

- (`init`) три выборки (для 3, 10 и 150 пользователей) для первичного изучения данных. Эти выборки во многом наследуют формат мастер-датасета: запросы пользователей сгруппированы в файлы (по одному на каждого пользователя) с объектами, содержащими метку времени и URL сайта.
- (`kaggle`) выборка подготовленная для основной части проекта в форме соревнования на Kaggle. Эта выборка имеет отличный от мастер-датасета формат: она разбита на две части (обучающую и проверочную) и содержит вектора из 20 признаков, сформированному по принципу описанному далее. Целевая переменная подготовлена для решения задачи One-vs-All: все объекты относящиеся к одному пользователю («Элис») обозначены 1, объекты всех прочих пользователей — 0.

- (`vw`) выборка из 400 пользователей, для обучения многоклассового классификатора при помощи пакета Vowpal Wabbit. Формат этой выборки аналогичен выборке `kaggle` за исключением целевой переменной — она содержит id пользователя которому принадлежит сессия.

Задача проекта — построить модель и обучить классификатор для определения принадлежности сессий пользователю Элис, который побъёт заданные бэйзлайны в соревновании на kaggle.

# 2 Предобработка данных

Первым этапом работы над проектом являлось изучение исходных данных. Изучен источник данных и его характерные особенности, итоги изложены в разделе 1.

Несколько строк содержимого мастер-датасета:

```
0001;15-11-2013T07:52:26;www.google.fr
0001;15-11-2013T07:52:27;www.gstatic.com
0001;15-11-2013T07:52:27;ssl.gstatic.com
0001;15-11-2013T07:52:28;apis.google.com
0001;15-11-2013T07:52:28;www.google.com
0001;15-11-2013T07:52:29;www.google.com
0001;15-11-2013T07:52:37;www.gstatic.com
0001;15-11-2013T07:52:37;ssl.gstatic.com
0001;15-11-2013T07:52:37;apis.google.com
0001;15-11-2013T07:52:37;www.google.fr
```

В отличие от матер-датасета, где все объекты в одном файле, в выборке `init` объекты разных пользователей разбиты по отдельным файлам. В остальном выборки идентичны. Вот несколько строк одного из файлов выборки `init`:

```
timestamp,site
2013-11-15 08:01:09,www.google.fr
2013-11-15 08:01:10,www.google.fr
2013-11-15 08:01:11,apis.google.com
2013-11-15 08:01:12,www.google.com
2013-11-15 08:01:16,www.google.fr
2013-11-15 08:01:16,apis.google.com
2013-11-15 08:01:17,www.google.fr
2013-11-15 08:01:17,www.google.com
2013-11-15 08:02:06,cnfg.toolbarservices.com
```

Продемонстрируем идентичность содержимого выборок на примере пользователя с $ID = 6$:

```
import pandas as pd

master_ds = pd.read_csv(
    "./dispoSite/allcats/cat0006.csv",
    sep=";",
    header=None,
    parse_dates=[1],
    dayfirst=True,
    names= ["userid", "timestamp", "site"],
)

init_ds = pd.read_csv(
    "./capstone_user_identification/150users/user0006.csv",
    parse_dates=["timestamp"]
)

master_ds = master_ds.groupby("timestamp")["site"].agg([lambda x: set(x)])
init_ds = init_ds.groupby("timestamp")["site"].agg([lambda x: set(x)])

equal = "Да" if (master_ds == init_ds).all().all() else "Нет"
print(f"Содержимое одинаковое? {equal}")
```

```
Содержимое одинаковое? Да
```

Далее, для определения пользователя по посещаемым сайтам, нам требуется разбить последовательность посещений сайтов на подпоследовательности, и из этих подпоследовательностей вычленить паттерны. По характерности этих паттернов для классифицируемого пользователя можно принять решение о принадлежности сессии пользователю.

Мы располагаем только URL сайтов и временем их посещений. Поэтому пользователь будет классифицироваться на основе паттернов посещений различных сайтов и времени этих посещений. Например пользователь может редко заходить на сайты определённой тематики, или для него характерна некоторая конкретная последовательность посещений сайтов. Временные признаки будут включать в себя такие вещи как время суток запроса, длительность сессии и прочее.

Из сказанного следует следующее:

- для нашей задачи важна упорядоченность данных по времени;
- работать с отдельными запросами не интересно, требуется последовательность запросов;
- для работы с такими последовательностями необходимо векторизовать их (например bag-of-word, или Tf-Idf);
- необходимо определить метод разбиения потока запросов пользователя на подпоследовательности («сессии»).

Разбивку на сессии делаем методом скользящего окна. Для начала составляем словарь сайтов, для отображения URL сайта в ID. Затем используя словарь считываем данные и загружаем их в датафрейм. Далее листинг соответствующих функций:

```
from glob import glob
import os
from tqdm.auto import tqdm
```

```
from collections import defaultdict

def prepare_dict(path_to_csv_files):

    file_list = sorted(glob(os.path.join(path_to_csv_files, "user*.csv")))

    fr_dict = defaultdict(int)
    for file_name in tqdm(file_list):
        with open(file_name, "r") as file:
            file.readline()
            for l in file:
                fr_dict[ l.split(",")[1][:-1] ] += 1
    fr_dict = sorted(fr_dict.items(), key=lambda x: x[1], reverse=True)
    fr_dict = {site: (i, freq) for i, (site, freq) in enumerate(fr_dict, 1)}

    return fr_dict
```

```
from more_itertools import windowed
import numpy as np

def prepare_dataset(
    path_to_csv_files, fr_dict, session_length=10, window_size=10
):

    file_list = sorted(glob(os.path.join(path_to_csv_files, "user*.csv")))

    sessions = []
    for file_name in tqdm(file_list):
        base_file_name = os.path.basename(file_name)
        user_id = int(base_file_name[len("user") : -len(".csv")])

        with open(file_name, "r") as file:
            file.readline()
            for fold in windowed(file, session_length, step=window_size):

                timestamps, sites = list(
                    zip(*[rec[:-1].split(",") for rec in fold if rec])
                )
                sites = [fr_dict[site][0] for site in sites]

                session = [0] * len(sites) * 2
                session[::2] = sites
                session[1::2] = timestamps
                session += [0] * (session_length * 2 - len(session))
                session.append(user_id)

                sessions.append(session)
```

```
    feature_names = [0] * session_length * 2
    feature_names[::2] = [f"site{i}" for i in range(1,session_length+1)]
    feature_names[1::2] = [f"time{i}" for i in range(1,session_length+1)]
    feature_names.append("target")
    out = pd.DataFrame(sessions, columns=feature_names)

    dtfmt = "%Y-%m-%d %H:%M:%S"
    time_cols = [f"time{i}" for i in range(1,session_length+1)]
    out[time_cols] = (
        out[time_cols]
        .replace(0, np.nan)
        .apply(lambda x: pd.to_datetime(x, format=dtfmt))
    )
    return out
```

Продемонстрируем первые строки полученного датафрейма:

| | site1 | time1 | ... | site10 | time10 | target |
|---|---|---|---|---|---|---|
| **0** | 192 | 2013-11-15 08:12:07 | … | 203 | 2013-11-15 08:12:40 | 31 |
| **1** | 415 | 2013-11-15 08:12:40 | … | 217 | 2013-11-15 08:17:16 | 31 |
| **2** | 55 | 2013-11-15 08:17:25 | … | 897 | 2013-11-15 08:21:43 | 31 |
| **3** | 473 | 2013-11-15 08:21:43 | … | 199 | 2013-11-15 08:22:13 | 31 |
| **4** | 342 | 2013-11-15 08:22:13 | … | 675 | 2013-11-15 08:22:19 | 31 |
| | | … | | | … | |

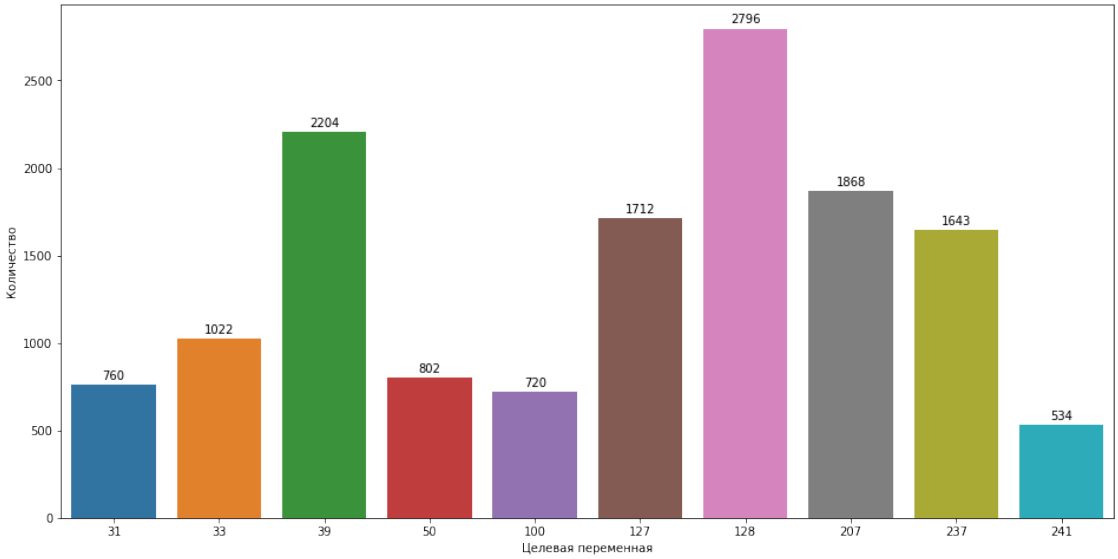
Дополнительно реализована функция конвертации признаков *«siteN»* в BoW, листинг далее:

```
from scipy.sparse import csr_matrix

def make_csr(X):
    row,col,val = [],[],[]
    for i, r in enumerate(np.array(X)):
        for id,freq in Counter(r).items():
            row.append(i)
            col.append(id)
            val.append(freq)
    return csr_matrix((val, (row, col)))[:,1:]
```

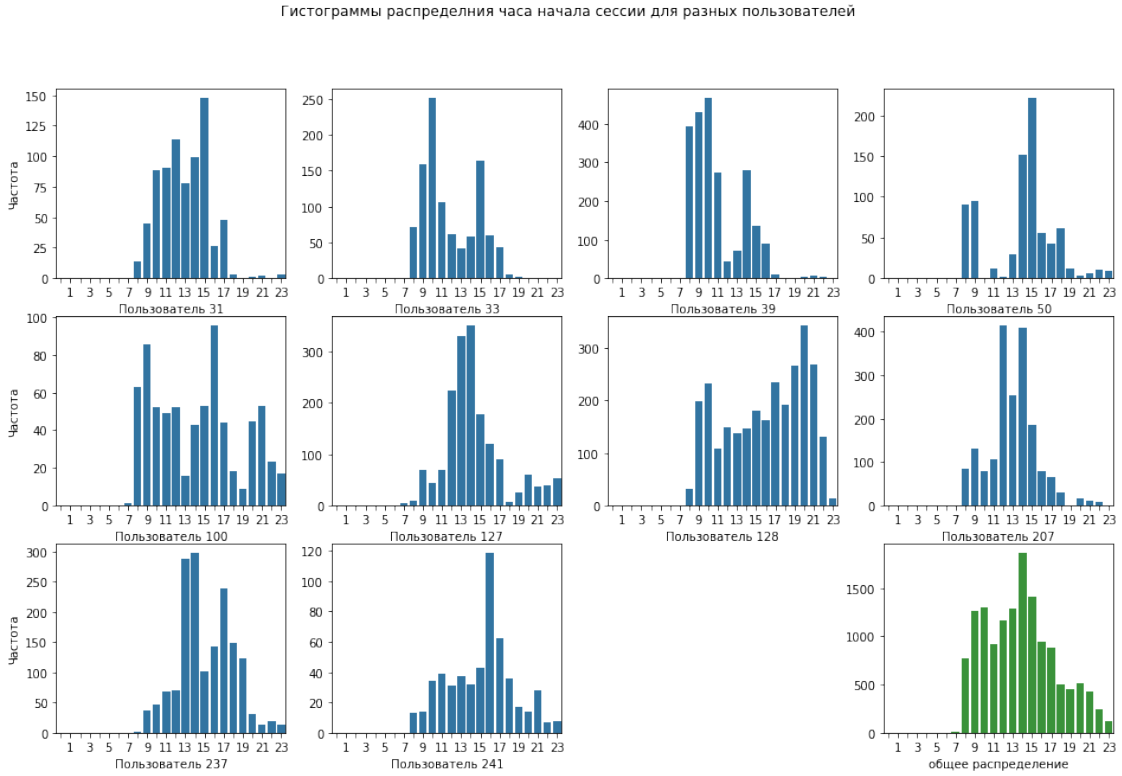
# 3 Анализ признаков

Далее проведено исследование данных и подбор новых признаков. Исследование проводилось на `init`-выборке 10 пользователей.

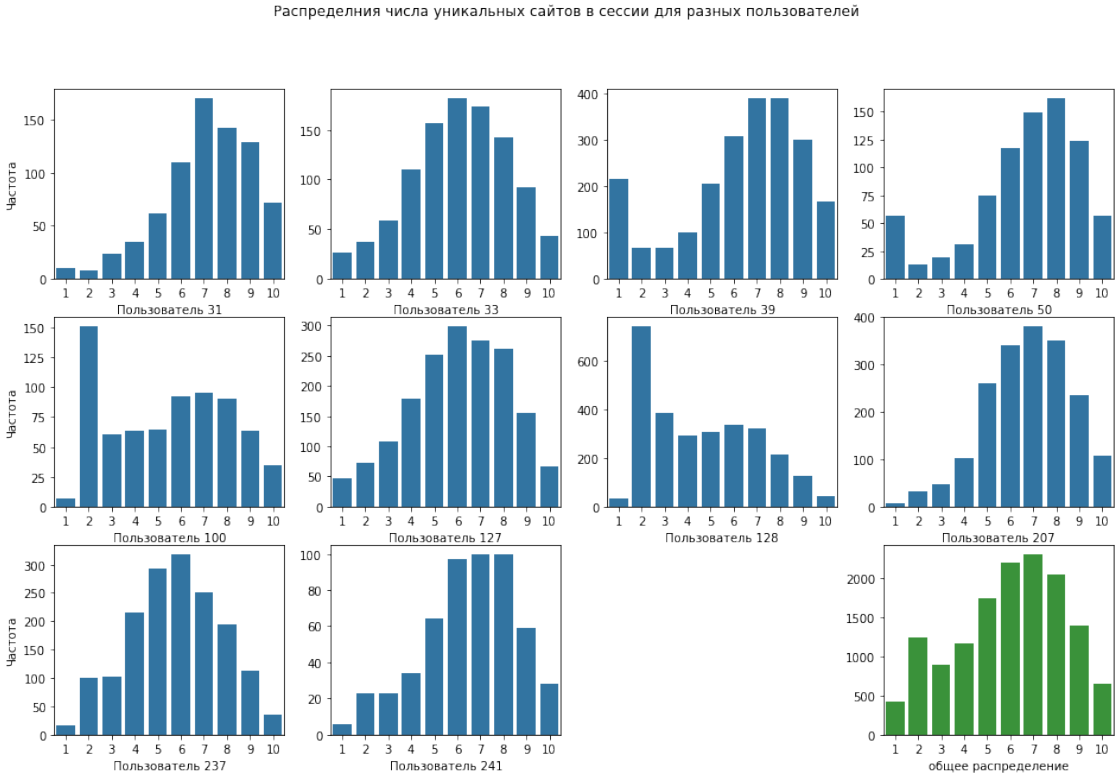
- Распределение целевого класса в `init` выборке смещённое. Это отражает то же свойство матер-датасета: для разных пользователей разное количество запросов. Дополнительно можно увидеть что при классификации One-vs-All классы будут очень несимметричными:



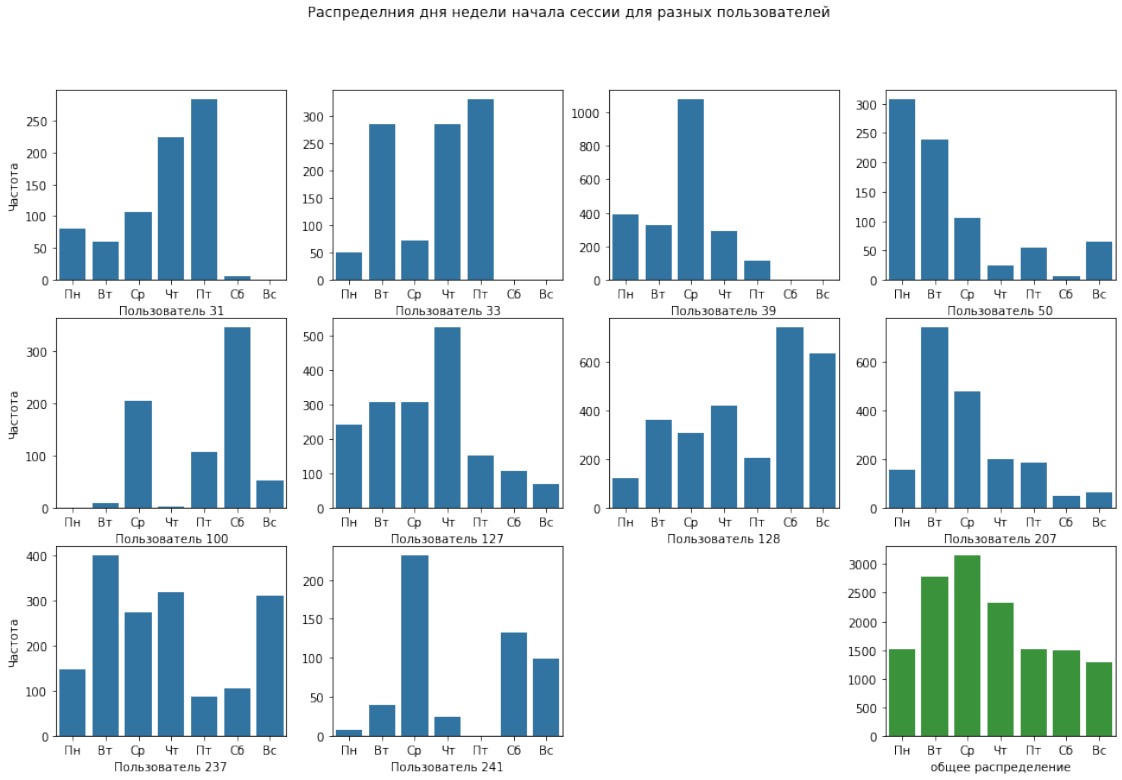
- Изучено распределение времени начала сессии. В генеральной совокупности видно две моды, в 10 и в 14 часов. Условные распределения по пользователям демонстрируют вариативность по величинам и смещениям мод. Дополнительно, у некоторых пользователей проявляется третья, «вечерняя», не видимая в генеральной совокупности, мода (напр. у пользователя 241 в 21 час) . Т.о. этот признак может быть информативен и повышать качество модели:



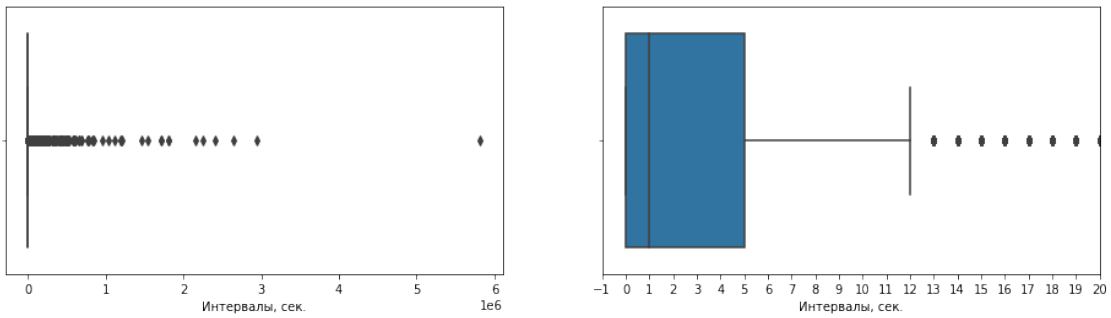
- Изучено распределение числа уникальных сайтов в сессии. Распределение в генеральной совокупности двухмодовое и смещено вправо. Условные распределения по пользователям схожи между собой. У некоторых пользователей присутствуют выбросы для 1-2 уникальных сайтов в сессии, именно эти выбросы похоже и дают вторую (правую) моду в распределении генеральной совокупности. Вероятно, этот признак будет хорошо работать только если в схеме One-vs-all целевой пользователь будет иметь большое количество сессий с одним-двумя уникальными сайтами. В прочих случаях ожидается слабое влияние этого признака:



- Изучено распределение дней начала сессии по пользователям. Видно что у каждого пользователя свой график работы, что приводит к разным распределениям для разных пользователей. Поэтому распределение генеральной совокупности похоже на равномерное, за исключением середины недели, где наблюдается большая частота сессий (вт-чт). Высокая вариативность условных распределений дня недели начала сессии по пользователям позволяет предположить что этот признак будет повышать качество классификации:



- Изучено влияние интервалов между запросами в сессии. Интервалы между запросами могут характеризовать «манеру сёрфинга» пользователя. Например, после поиска в поисковике — кто-то открывает сразу несколько найденных поисковой системой страниц и только потом по очереди их читает, а кто-то открывает и читает по одной. Другой пример: некоторые пользователи могут чаще посещать странички с длинными статьями (лонгриды), а кто-то — с короткими заметками и чаще переходить на новые страницы. На левом графике ниже построен «ящик с усами» для интервалов между запросами; правый график — увеличение левого на области $[0, 20]$:




Численные значения в таблице ниже:

| Статистика | Значение |
|---:|---:|
| Максимум | 5812677 |
| Квантиль уровня 0,99 | 705 |
| Квантиль уровня 0,95 | 88 |

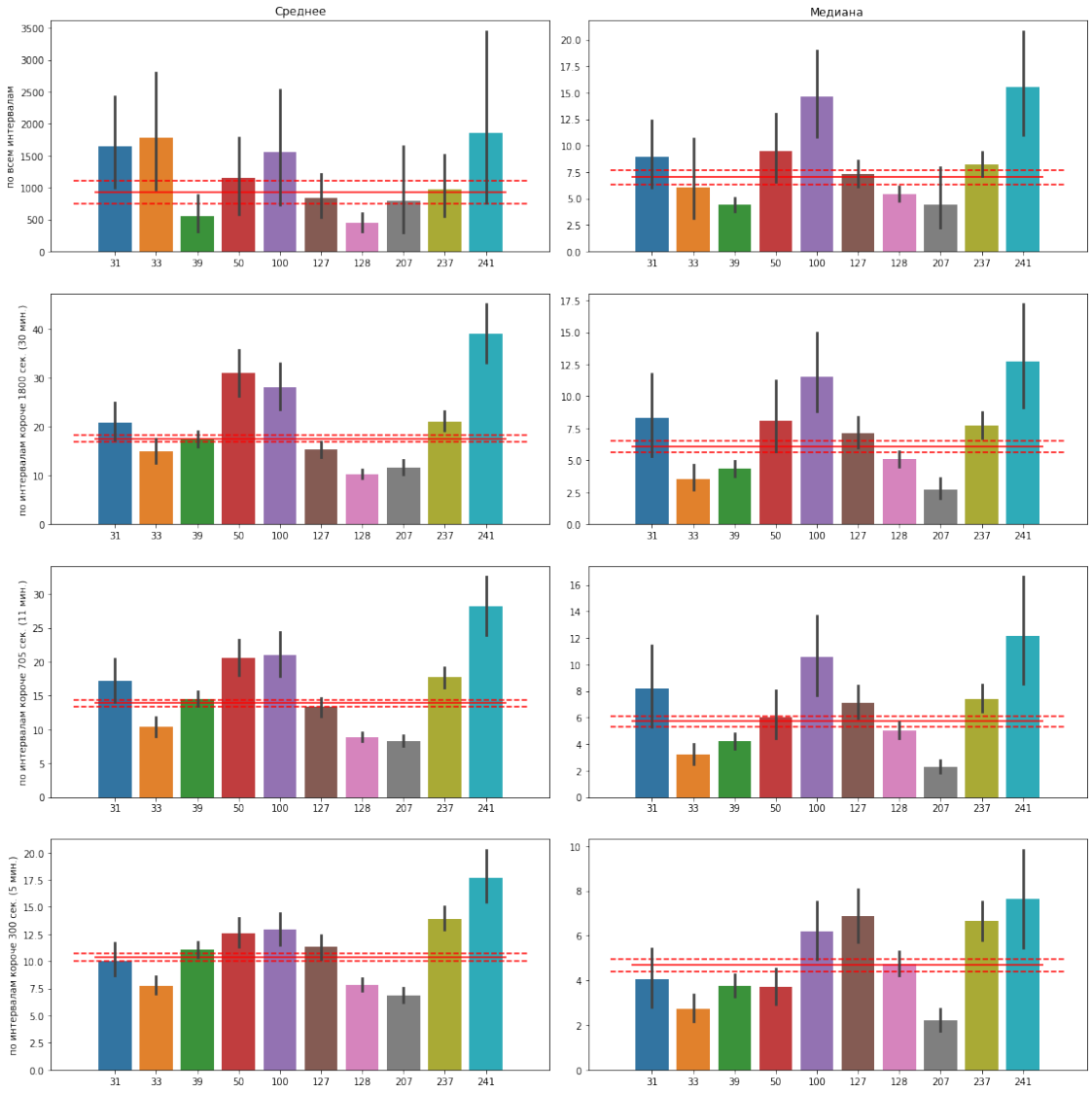
На основе диаграммы «ящик с усами» сделан вывод, что основная масса интервалов распределена между 0 и 705 секундами (примерно 11 минут), а многочисленные выбросы - это периоды неактивности пользователя попавшие в сессии вследствие выбранного способа разделения (т.е. «реальные» границы сеансов пользователя). Исходя из этого была предпринята попытка поиска оптимальный границы отсечений длинных интервалов, для выявления характерных для конкретных пользователей закономерностей.

На графике ниже отражены средние значение стаистик (среднее и медиана) интервалов между запросами по пользователям. Чёрные линии отражат доверительные интервалы для этих статистик. Горизнтальные красные линии соотетствуют среднему и доверительному интервалу статистики в генеральной совокупности. Этот график демонстрирует следующие соотношения доверительныз интервалов статистик по пользователям к стаистикам по генеральной совокупности:

- для всех интервалов - 7 из 10 (среднее) и 6 из 10 (медиана): это значит что среднее интервалов плохо дифференцирует пользователей;
- при ограничении длины интервалов ситуация начинает улучшаться и уже при ограничении длинны интервалов 10-ю минутами 7 из 10 пользователей (6 из 10 для медианы)

имеют непересекающиеся доверительные интервалы с доверительным интервалом для генеральной совокупности;

- при дальнейшем ограничении длины интервала средние различия начинают уменьшаться.




Таким образом делается вывод, что для улучшения качества классификации разумно использовать интервальные статистики (матожидание и среднеквадратичное отклонение) для интервалов ограниченных сверху по длительности. Порог выбран длительностью в 705 сек.

- дополнительно исследованы признаки: число нулевых интервалов в сессии, максимальный интервал в сессии, число запросов по ip-адресу в сессии, число нетиповых (не из top-4) TLD для запросов в сессии. Эти признаки не показали однозначной корреляции с целевой переменной, поэтому решено было посмотреть на практике насколько они смогут улучшить качество классификации.

# 4 Пробные обучения основных алгоритмов

Далее были проведены пробные обучения алгоритмов «K ближайших соседей», «Случайный лес», логистической регрессии и метода опорных векторов с линейным ядром.

Для обучения использовалась выборка `init` для 10 пользователей. Признаки формировались разбиением на сессии описанным выше способом, далее сайты сессий векторизовались методом bag-of-word. Прочих признаков при этом не использовалось.

Для многоклассовой классификации лучшие результаты показали логистическая регрессия и линейный SVM, достигнув сопоставимых показателей точности (accuracy) 0,78 на отложенной выборке после подбора параметра регуляризации.

Так же были проверены разные длины сессий и разные ширины окна (при этом использовалась LinearSVM с подобранным ранее параметром регуляризации)

| длина сессии | ширина окна | CV точность | точность на отл. выборке |
|---|---|---|---|
| 15 | 10 | 0.824 | 0.840 |
| 10 | 10 | 0.767 | 0.781 |
| 15 | 7 | 0.850 | 0.854 |
| 10 | 7 | 0.798 | 0.807 |
| 7 | 7 | 0.755 | 0.762 |
| **15** | **5** | **0.867** | **0.875** |
| 10 | 5 | 0.818 | 0.825 |
| 7 | 5 | 0.773 | 0.785 |
| 5 | 5 | 0.725 | 0.736 |

А ниже — результаты классификации на `init` выборке для 150 пользователей:

| длина сессии | ширина окна | CV точность | точность на отл. выборке |
|---|---|---|---|
| 15 | 10 | 0.549 | 0.575 |
| 10 | 10 | 0.463 | 0.484 |
| 15 | 7 | 0.583 | 0.609 |
| 10 | 7 | 0.502 | 0.524 |
| 7 | 7 | 0.437 | 0.453 |
| **15** | **5** | **0.614** | **0.636** |
| 10 | 5 | 0.527 | 0.546 |
| 7 | 5 | 0.465 | 0.482 |
| 5 | 5 | 0.408 | 0.422 |

Видно, что хотя точность при 150 пользователях существенно снижается, соотношения точностей между собой почти не меняются, и наибольшую точность опять даёт разбиение 15-5.

# 5 Выбор основной модели и обучение классификатора

Дальнейшая работа по проекту велась в форме соревнования на kaggle. Для этого соревнования составителями была сформирована выборка `kaggle` на основе мастер-датасета. Метод формирования аналогичен использованному нами, с параметрами длина сессии 10 сайтов, ширина окна — 10 сайтов. Дополнительно введено ограничение в 30 минут на общую продолжительность сессии. То есть сессия считается оконченной либо когда пользователь посетил 10 сайтов подряд, либо когда сессия заняла по времени более 30 минут.

В соревновании решалась задача в постановке One-vs-All, а метрикой качества выступала площадь под ROC кривой.

## 5.1 Создание признаков

Для исследования моделей и их производительности, на основе исходных обучащей и тестовой выборок, сформированы датафреймы с признаками. Извлечение признаков объеденены в одну общую функцицю, листинг которой приведён ниже:

```
def fe(df_in, site_dict, tld_dict, url_to_tld_map, target=False):

    """
    признаки:
        site### - ID сайтов в сессии
        tld### - ID TLD сайтов

        tdiff### - интервал между соседними запросами
                    (между первым и вторым, 3 и 2 и т.п)

        tdiff_mean, tdiff_std, tdiff_median, tdiff_mad -
                    соответствующие статистики для tdiff###

        session_duration - продолжительность сессии в секундах

        n_unique_sites - число уникальных сайтов в сессии
        n_unique_tlds - число уникальных TLD в сессии
        n_ips - число запросов ip-адресов в сессии

        start_year - год начала сессии
        start_month - месяц начала сессии
        start_day - день начала сессии
        start_hour - час начала сессии
        start_minute - минуты начала сессии
        start_day_of_week - день недели начала сессии
        start_weekend - 1 если сессия началась в выходные
        start_year_month - комбинация года и месяца
        start_week_hour - комбинация недели и часа (час с начала недели)
        start_hour_minute - комбинация часов и минут (минуты с начала дня)
        start_epoch - время начала сессии в POSIX-формате
        start_morning, start_noon, start_evening - флаги времени суток начала сессии

        automatic - флаг сессии из серии запросов с равными интервалами одного сайта
        final - флаг финальной сессии (есть "нули")
    """

    # извлекаем ID сайтов в сессии
    scols = [c for c in df_in.columns if re.match(r"^site\d+$", c)]
    df_out = df_in[scols].fillna(0).astype(int)
```

```
    # извлекаем ID TLD сайтов в сессии
    tldcols = [f"tld{i}" for i in range(1,len(scols)+1)]
    df_out[tldcols] = df_out[scols].applymap(lambda x: url_to_tld_map[x])

    # Временные интервалы между запросами сайтов в сессии
    tcols = [c for c in df_in.columns if re.match(r"^time\d+$", c)]
    timestamps = df_in[tcols].apply(lambda x: pd.to_datetime(x, dayfirst=True))
    temp = timestamps.to_numpy()[:, 1:] - timestamps.to_numpy()[:, :-1]
    tdcols = [f"tdiff{i}" for i in range(1, len(tcols))]
    temp = pd.DataFrame(temp, columns=tdcols, index=timestamps.index)
    temp = temp.astype(np.timedelta64(1, "s"))
    df_out[tdcols] = temp.fillna(0).astype(int)

    # статистики интервалов между запросами сайтов в сессии
    temp = df_out[tdcols]
    df_out[f"tdiff_mean"] = temp.mean(axis=1)
    df_out[f"tdiff_median"] = temp.median(axis=1)
    df_out[f"tdiff_std"] = temp.std(axis=1)
    df_out[f"tdiff_mad"] = temp.apply(
        lambda x: abs(x - df_out[f"tdiff_median"])
    ).median(axis=1)

    # статистики интервалов между запросами сайтов в сессии
    temp = df_out[tdcols][df_out[tdcols] <= 705]
    df_out[f"tdiff_short_mean"] = temp.mean(axis=1)
    df_out[f"tdiff_short_median"] = temp.median(axis=1)
    df_out[f"tdiff_short_std"] = temp.std(axis=1)
    df_out[f"tdiff_short_mad"] = temp.apply(
        lambda x: abs(x - df_out[f"tdiff_short_median"])
    ).median(axis=1)

    # число уникальных сайтов, TLD в сессии
    df_out["n_unique_sites"] = df_in[scols].apply(
        lambda row: len(set([x for x in row if not np.isnan(x)])), axis=1
    )
    df_out["n_unique_tlds"] = df_out[tldcols].sum(axis=1)

    # число обращений по IP в сессии
    ipv4 = r"^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?
↪[0-9][0-9]?)$"
    ipv6 = r"^\s*((([0-9A-Fa-f]{1,4}:){7}([0-9A-Fa-f]{1,4}|:))|(([0-9A-Fa-f]{1,4}:){6}(:
↪[0-9A-Fa-f]{1,4}|((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:
↪))|(([0-9A-Fa-f]{1,4}:){5}(((:[0-9A-Fa-f]{1,4}){1,2})|:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.
↪(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|(([0-9A-Fa-f]{1,4}:){4}(((:
↪[0-9A-Fa-f]{1,4}){1,3})|((:[0-9A-Fa-f]{1,4})?:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.
↪(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3})|:))|(([0-9A-Fa-f]{1,4}:){3}(((:
↪[0-9A-Fa-f]{1,4}){1,4})|((:[0-9A-Fa-f]{1,4}){0,2}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.
↪(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|(([0-9A-Fa-f]{1,4}:){2}(((:
↪[0-9A-Fa-f]{1,4}){1,5})|((:[0-9A-Fa-f]{1,4}){0,3}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.
↪(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|(([0-9A-Fa-f]{1,4}:){1}(((:
↪[0-9A-Fa-f]{1,4}){1,6})|((:[0-9A-Fa-f]{1,4}){0,4}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.
↪(25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)){3}))|:))|(:(((:[0-9A-Fa-f]{1,4}){1,7})|((:
↪[0-9A-Fa-f]{1,4}){0,5}:((25[0-5]|2[0-4]\d|1\d\d|[1-9]?\d)(\.(25[0-5]|2[0-4]\d|1\d\d|[1-9]?
↪\d)){3}))|:)))(%.+)?\s*$"
    ipre = f"({ipv4})|({ipv6})"
    ip_ids = [id for url, id in site_dict.items() if re.match(ipre, url)]
    df_out["n_ips"] = df_in[scols].isin(ip_ids).sum(axis=1)

    # признаки извлекаемые из даты и времени начала сессии
    df_out["start_date"] = timestamps["time1"]
    df_out["start_year"] = timestamps["time1"].dt.year
    df_out["start_month"] = timestamps["time1"].dt.month
    df_out["start_day"] = timestamps["time1"].dt.day
    df_out["start_hour"] = timestamps["time1"].dt.hour
    df_out["start_minute"] = timestamps["time1"].dt.minute
```

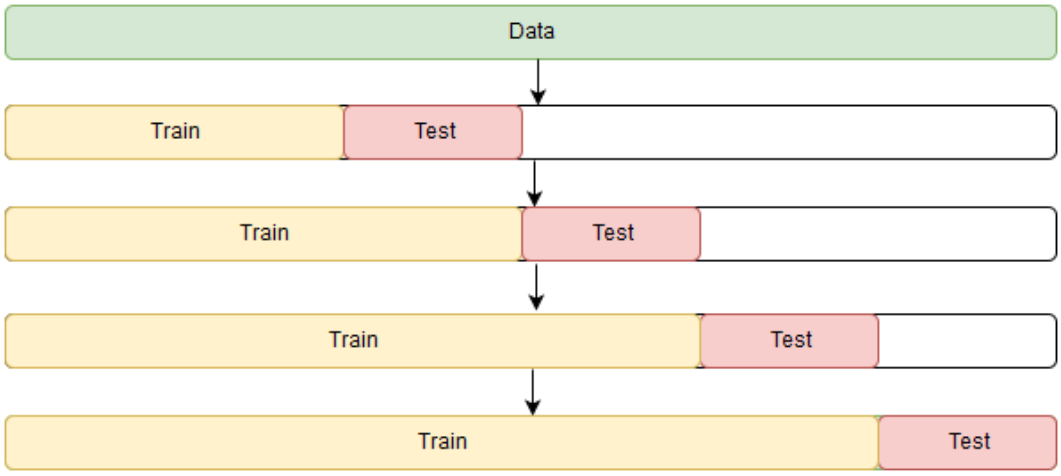
```
    df_out["start_day_of_week"] = timestamps["time1"].apply(lambda x: x.weekday())
    df_out["start_weekend"] = (df_out["start_day_of_week"] >= 5).astype("int")

    df_out["start_year_month"] = df_out["start_year"] * 100 + df_out["start_month"]
    df_out["start_week_hour"] = df_out["start_day_of_week"] * 24 + df_out["start_hour"]
    df_out["start_hour_minute"] = df_out["start_hour"] * 60 + df_out["start_minute"]
    df_out["start_epoch"] = timestamps["time1"].astype("int64") // 1e9

    # флаги времени суток начала сессии
    df_out[f"start_morning"] = (
        (df_out["start_hour"] >= 6) & (df_out["start_hour"] <= 11)
    ).astype("int")
    df_out[f"start_noon"] = (
        (df_out["start_hour"] >= 12) & (df_out["start_hour"] <= 17)
    ).astype("int")
    df_out[f"start_evening"] = (
        (df_out["start_hour"] >= 18) & (df_out["start_hour"] <= 23)
    ).astype("int")

    # продолжительность сессии
    df_out["session_duration"] = (
        (timestamps.max(axis=1) - timestamps.min(axis=1))
        .apply(lambda x: x.total_seconds())
        .astype(int)
    )

    # флаг сессии из серии запросов с равными интервалами одного сайта
    df_out["automatic"] = (
        (df_out["tdiff_std"] == 0)
        & (df_out["n_unique_sites"] == 1)
        & ((df_out[tdcols] > 0).all(axis=1))
    ).astype(int)

    # флаг финальной сессии (есть "нули")
    df_out["final"] = (df_out[scols] == 0).any(axis=1).astype('int')

    if target:
        df_out["target"] = df_in["target"]
    return df_out
```

Особое внимание уделено тому, что бы каждый новый признак объекта формировался только на основе прочих признаков того же объекта, что бы не было «утечки» информации между объектами и переобучения модели.

Исходная тренировочная выборка сортируется по дате начала сессии и к ней применяется функция извлечения признаков. К исходной тестовой выборке функция применяется без предварительной сортировки. Дальнейшая работа ведётся с полученными датафреймами (fe_train и fe_test)

## 5.2 Схема валидации моделей

При решении задачи для валидации моделей использовался метод кроссвалидации. Так как данные имеют хронологический порядок, для разбиения выборки на фолды использовась схема TimeSeriesSplit с 10 разбиениями. При данной схеме на первом разбиении береётся первые 1/11 часть выборки для обучения, и вторая 1/11 часть — для валидации, на втором фолде для обучения — первые две 1/11 части и 3-ая — для валидации т.д. Риснуок ниже наглядно демонстрирует схему:

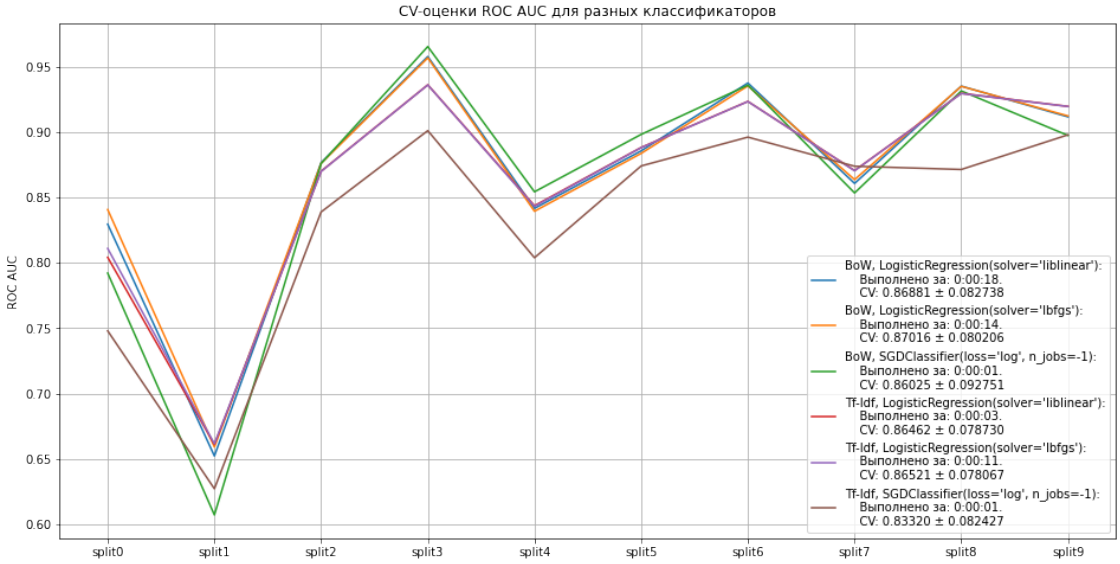



## 5.3 Выбор алгоритма классификации

Для векотризации сайтов в сессиях используются два класса из библиотеки scikit-learn: `CountVectorizer` и `TfidfVectorizer`

Ранее было показано, что на предварительных прогонах лучшие результаты показали логистическая регрессия и линейный SVM. Так как SVM в данной задаче медленней логистической регрессиии, остановимся на последней.

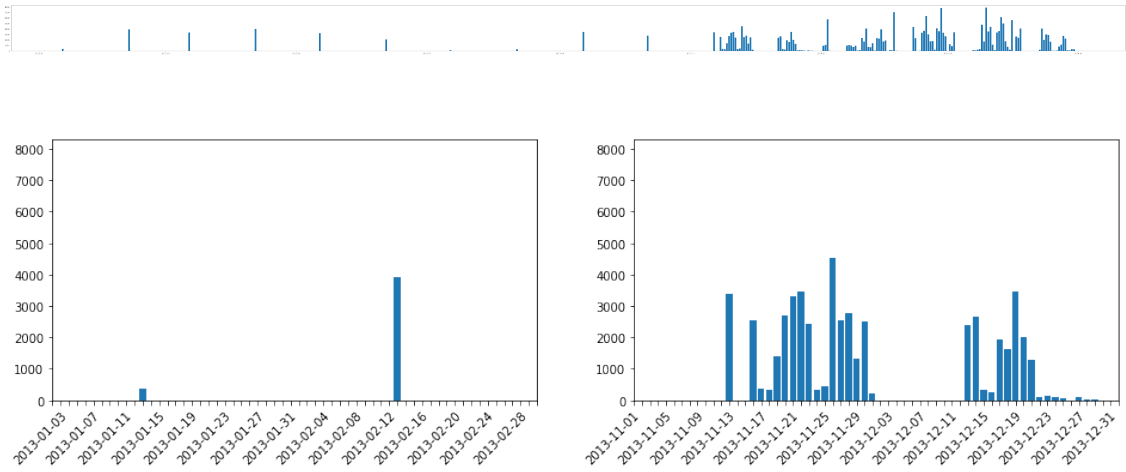
Далее приведены результаты теста разных методов оптимизации, поддерживаемых классом `LogisticRegression` библиотеки scikit-learn. Дополнительно был проверен метод стохастического градиентного спуска с логистической функцией потерь.




При сопоставимом качестве, выбран наиболее быстрый классификатор — логистическая регрессия с методом liblinear и Tf-Idf в качестве метода векторизации.

Так же видно, что первые два фолда по всем прогонам сильно отличаются от остальных. Так как мы генерируем фолды по схеме TimeSeriesSplit с 10 фолдами, первые для фолда содержат по 23051 и 46102 объектов соответственно. Вектора BoW/Tf-Idf содержат 50000 или 100000 признаков (при данном тесте использовался параметр $max\_features = 100000$). Поэтому валидация на первом и втором прогонах делалась в условиях, когда классификатор обучался на выборке с намного меньшей мощностью чем размерность признакового пространства. Поэтому классификатор мог оказаться недообучен.

Так же сделано другое наблюдение. Просадка в CV-скоринге наблюдается на 1,4, 7 и 9 разбиениях (для других способов векторизации наблюдается идентичная картина). В попытке найти что необычного появляется в валидационных фолдах на этих разбиениях, обнаружена ошибка в данных. Ниже показан график количества сессий по дате начала сессии:




Верхняя диаграмма отражает весь интервал с 01.01.2013 по 30.04.2014, две нижние - увеличенные участки верхней диаграммы.

Видно, что в выборке есть сессии начинающиеся в месяцах с января по ноябрь только 12 числа, далее идут сессии с 15 по 30 ноября, и в декабре сессий с 01 по 12 включительно нет. Такие же провалы с 1 по 12 числа есть и в каждом месяце 2014.

Так как данные для мастер-датасета собирались с ноября 2013 по май 2014 — сессий начинающихся в январе 2013 быть не может.

Характер распределения количества сессий на интервале с 01.01.2013 по 12.11.2013 позволяет предположить, что организаторами соревнования при подготовке обучающей и проверочной выборок допущена ошибка при парсинге дат, и даты с числами с 1 по 12 трактовались с переменой местами дня и месяца. Т.е. например дата 1 декабря 2013 (01.12.2013) некорректно распарсилась как 12 января (12.01.2013). Отсюда и сессии в месяцах когда их быть не может, и пропуски первых 12 дней в месяце. Схожая проблема обсуждается, например, тут: «Python Pandas : pandas.to_datetime() is switching day & month when day is less than 13»

Таким образом, сделано предположение что первые два фолда сильно зашумлены, т.к объём обучающей выборки мал и нарушен порядок сессий. Отсутствующими объектами выборок

в поздние месяцы можно объяснить менее значительные провалы CV-оценок на разбиениях 4,7 и 9.

Были предприняты попытки исправить датасет, обратно переставив местами день и месяц для дат где день меньше или равен 12, но т.к. в 2014 году произошло смешение дат для месяцев с небольшими порядковыми номерами между обучающей и проверочной выборками, корректную последовательность восстановить удалось только для 2013 года. Поэтому принято решение отказаться от корректировки.

В связи с искажением информации о датах для некоторых сессий, можно ожидать что признаки захватывающие информацию о дате начала сессий будут работать хуже (например, день недели начала сессии и т.п.).

Так как выбор гиперпараметров моделей и общая оценка производительности моделей зависит от кроссвалидации, а два первых разбиения сильно зашумлены, принято решение при кросс-валидации опускать оценки для первых двух разбиений.

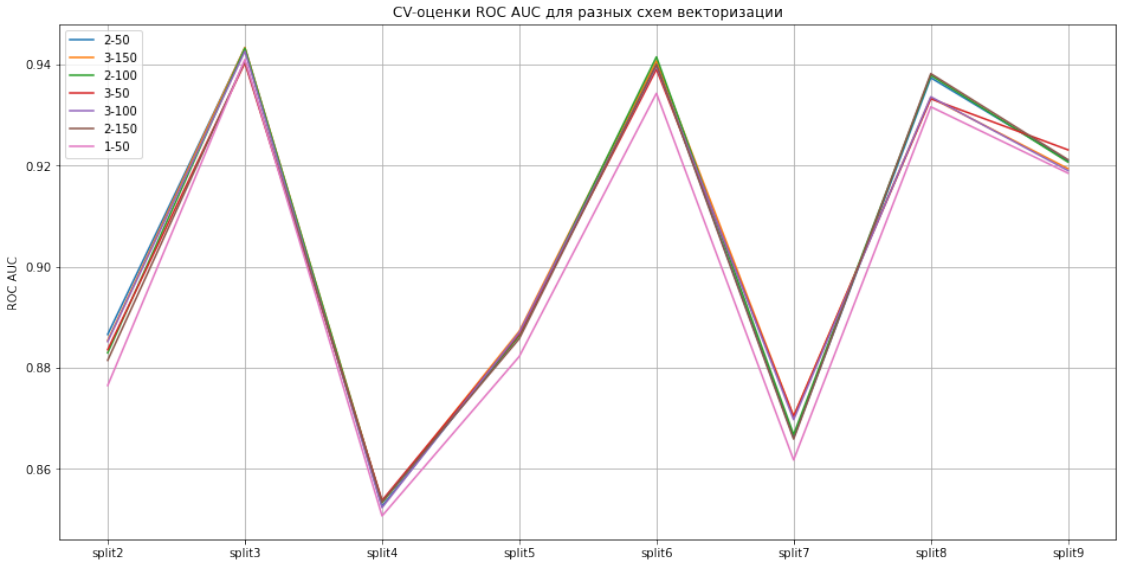
Ниже приведена таблица, показывающая дисперсию (СКО) метрики ROC AUC при отбрасывании первых n разбиений:

| | BoW, LR(liblinear) | BoW, LR(lbfgs) | BoW, SGD(log) | Tf-Idf, LR(liblinear) | Tf-Idf, LR(lbfgs) | Tf-Idf, SGD(log) |
|---|---|---|---|---|---|---|
| **drop 0** | 0.082738 | 0.080206 | 0.092751 | 0.078730 | 0.078067 | 0.082427 |
| **drop 1** | 0.086114 | 0.083908 | 0.092428 | 0.080222 | 0.080049 | 0.081418 |
| **drop 2** | 0.038430 | 0.038200 | 0.038407 | 0.031922 | 0.032016 | 0.031178 |
| **drop 3** | 0.039896 | 0.039645 | 0.040133 | 0.032246 | 0.032354 | 0.030959 |
| **drop 4** | 0.036073 | 0.035880 | 0.033238 | 0.031324 | 0.031443 | 0.031292 |
| **drop 5** | 0.029464 | 0.028394 | 0.030311 | 0.022887 | 0.022907 | 0.011613 |
| **drop 6** | 0.030881 | 0.029195 | 0.033488 | 0.023510 | 0.023523 | 0.012048 |
| **drop 7** | 0.031060 | 0.029766 | 0.034206 | 0.025810 | 0.025821 | 0.011760 |
| **drop 8** | 0.011761 | 0.011313 | 0.022665 | 0.005001 | 0.004964 | 0.013022 |

При отбрасывании первых двух фолдов оценка среднего становится более стабильной.

## 5.4 Выбор схемы векторизации

Далее проверены разные схемы векторизации Tf-Idf, а именно проверены гиперпараметры «максимальное число признаков» (50 тысяч и 100 тысяч) и «максимальная длинна n-грамм» (1, 2 и 3). Ниже показаны результаты кроссвалидации:




В легенде первое число — длина n-грамм, второе — максимальное количество признаков в тысячах. Видно, что (за исключением схемы 1-50) все схемы дают схожие оценки на кроссвалидации. Поэтому выбор гиперпараметра делался исходя из скорости обучения. Выбрана схема Tf-Idf 3-50.

## 5.5 Настройка параметра регуляризации

Поиск оптимального коэффициента регуляризации осуществлялся методом поиска по сетке. Для автоматизации поиска по сетке реализована итеративная процедура поиска C:

1. ищется оптимальное $C$
2. если оптимальное C на границе сетки —- смещаем границы сетки в соответствующую сторону и переходим на шаг 1
3. если оптимальное $C$ не на границе сетки:
   3.1. если разность оценок справа и слева от $C$ (в сетке) менее порога — считаем нашли оптимальное $C$
   3.2. иначе делаем новую сетку с границами равным правому и левому соседям текущего оптимального $C$ и переходим на шаг 1

Далее приведена реализация функции:

```
def lrcv(lr, rng, cv, X, y, tol=1e-4, max_iter=5, skip_first=2):

    """
    lr - регрессор
    rng - границы и шаг для сетки поиска
    cv - генератор индексов для кроссвалидации
    X,y - обучающая выборка и целевая переменная
    tol - криетрий остановки поиска
    max_iter - максимальное количество итераций
    skip_first - сколько первых фолдов пропустить при
                 оценке оптимального значения параметра
    """

    with tqdm.tqdm(total=max_iter) as progress_bar:
        for i in range(max_iter):

            pg = {"C": np.linspace(*rng)}
            if pg["C"][0] == 0:
                pg["C"] = pg["C"][1:]
            gscv = GridSearchCV(
                lr,
                param_grid=pg,
                cv=cv,
                scoring="roc_auc",
                n_jobs=-1,
                refit=False,
                return_train_score=True,
            ).fit(X, y)

            cvr = pd.DataFrame(gscv.cv_results_).fillna(0)
            splits_cols = [c for c in cvr.columns if re.match(r"^split\d+_test_score$", c)]

            cvr["mean_test_score"] = cvr[splits_cols].iloc[:, skip_first:].mean(axis=1)
            cvr["std_test_score"] = cvr[splits_cols].iloc[:, skip_first:].std(axis=1)
            top = cvr["mean_test_score"].idxmax()

            if top > 0 and top < len(pg["C"]) - 1:
                diff = cvr["mean_test_score"][top + 1] - cvr["mean_test_score"][top - 1]
                if diff < tol:
                    progress_bar.total = i + 1
                    progress_bar.update(1)
                    break

            parms = cvr["param_C"]
            new_l = parms[top - 1] if top > 0 else cvr(parms[top] / len(pg["C"]))
            new_u = parms[top + 1] if top < len(pg["C"]) - 1 else parms[top] * len(pg["C"])
            rng = (new_l, new_u, len(pg["C"]))
            progress_bar.update(1)
        else:
            print(f"Достигнуто максимальное количество итераций ({max_iter}).")
            print("Вероятно оптимальное значение не найдено")

    cvr["skipped"] = skip_first
    splits_cols = [c for c in cvr.columns if re.match(r"^split\d+_((test)|(train))_score$", c)]
    other_cols = ["mean_test_score", "std_test_score", "params", "skipped"]
    return cvr.loc[top, splits_cols + other_cols]
```

## 5.6 Подбор дополнительных признаков

Далее проведено обучение на разных наборах дополнительных признаков («начала сессии», «интервальных» и прочих), и осуществлена оценка полученных моделей методом кроссвалидации. Методология обучения модели для данного набора дополнительных признаков при этом следующая:

1. берём Tf-Idf представление сайтов в сессиях (по схеме 3-50);
2. конкатенируем с набором дополнительных признаков, для которых не требуется предобработки;
3. конкатенируем с стандартизированным (класс `StandardScaler` библиотеки scikit-learn) набором дополнительных признаков, для которых требуется стандартизация;
4. конкатенируем с датафреймом, полученным «One-Hot» кодированием категориальных признаков из набора дополнительных признаков;
5. полученный датафрейм передаём для поиска оптимального $C$ в вышеописанную функцию lrcv().

Ниже листинг этой функции:

```
def cvtest_model(
    X,
    y,
    ft_df,
    ft_asis=None,
    ft_scale=None,
    ft_dummy=None,
    solver="liblinear",
    start_range=(0, 10, 11),
    skip_first=2,
):
    """
    ft_df - датафрейм с признаками
    ft_asis - имена признаков, не требующих обработки
    ft_scale - имена признаков, которые надо нормализовать
    ft_dummy - имена признаков, которые надо закодировать OHE
    """

    X_ = X.copy()

    if ft_asis is not None:
        X_ = sparse.hstack((X_, ft_df[ft_asis])).tocsr()

    if ft_scale is not None:
        ft_scale = StandardScaler().fit_transform(ft_df[ft_scale])
        X_ = sparse.hstack((X_, ft_scale)).tocsr()

    if ft_dummy is not None:
        ft_dummy = pd.get_dummies(ft_df[ft_dummy], columns=ft_dummy, drop_first=True)
        X_ = sparse.hstack((X_, ft_dummy)).tocsr()

    clf = LogisticRegression(max_iter=1e5, solver=solver)
    return lrcv(clf, start_range, ts, X_, y, skip_first)
```

Результаты вызова этой функции записывались в словарь для последующего сравнения.

Среди признаков начала сессии наилучший результат показал признак «час начала сессии» трактуемый как категориальный (т.е. с применением к нему One-Hot кодирования). Добавление к нему признака «время суток начала сессии» немного улучшает среднюю оценку ROC AUC. Оценки ROC AUC при включении прочих признаков показывают худшие результаты. Вероятно основанные на датах признаки (день недели, выходной и др.) слишком зашумлены из-за ошибки с датами в исходных данных и поэтому не сильно улучшают оценку. В итоге принято решение включить в итоговую модель признаки «час начала сессии» и «время суток начала сессии», остальные же не включать. Ниже приведён график кроссвалидации отобранных признаков и, для сравнения пары, отброшенных признаков.


Таким же образом проверено влияние «интеравльных» признаков на качество классификации. В данном случае, при добавлении к модели признаков этого класса существенного улучшения классификации не происходит - рост средней по фолдам оценки ROC-AUC в лучшем случае не превышает $0,00125$.

Аналогичные результаты получены при проверке прочих признаков, при этом максимальное улучшение средней оценки ROC-AUC ещё ниже и не превышает $0,00024$.

## 5.7 Обучение итоговой модели

В итоговой модели используются следующие признаки:

- векторизованное представление сайтов в сессии, с следущими параметрами векторизации:
  - максимальная длина n-грамм: 3
  - максимальное число признаков: 50000
- час начала сессии, кодированный как категориальный методом «One Hot Encoding»
- бинарные признаки времени суток начала сессии (утро, день, вечер)

Классификатор обучался методом логистической регрессии с параметром регуляризации $C = 12$. При этом в качестве метода оптимизации использовался «L-BFGS», т.к. он даёт ответ немного более близкий к минимуму.

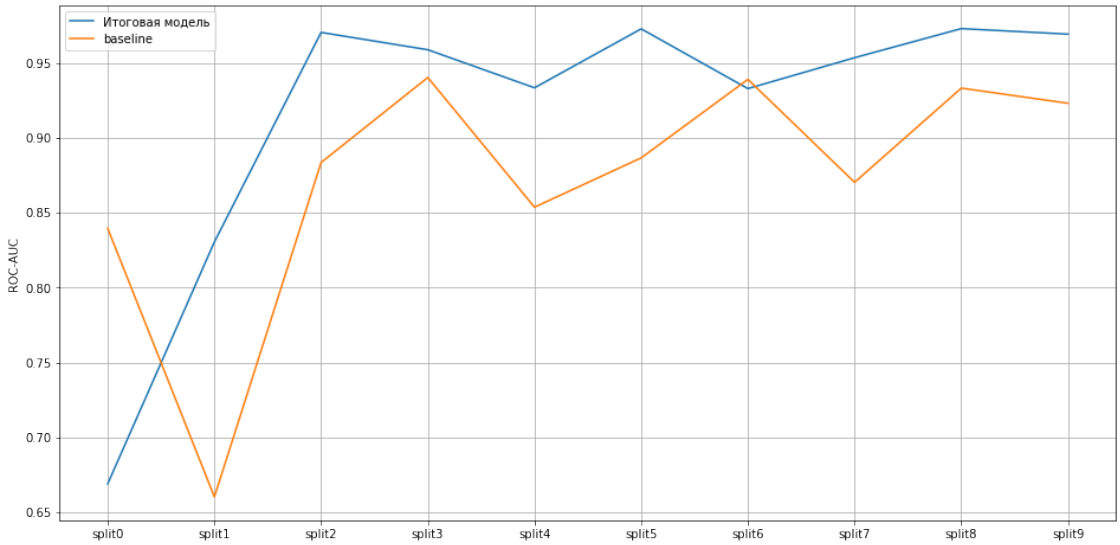
Коэффициенты обученной модели распределены следующим образом:

| | |
|---|---|
| Среднее | -0.0045 ± 0.6709 |
| минимум | -13.264 |
| .01 квантиль | -1.5868 |
| .05 квантиль | -0.6634 |
| .50 квантиль | -0.0241 |
| .95 квантиль | 1.0126 |
| .99 квантиль | 2.8480 |
| максимум | 13.7815 |

Коэффициенты при дополнительных признаках приведены в следующей таблице:

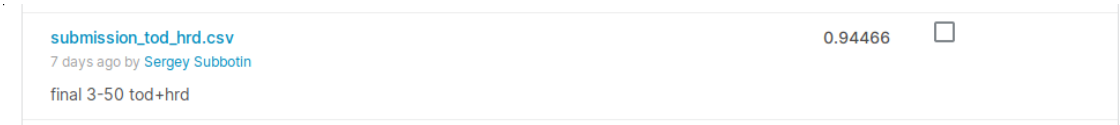
| | |
|---|---|
| start_morning | -2.102714 |
| start_noon | 2.176519 |
| start_evening | -0.193537 |
| start_hour_8 | -2.127341 |
| start_hour_9 | 2.506417 |
| start_hour_10 | -2.688524 |
| start_hour_11 | 0.286802 |
| start_hour_12 | 1.016875 |
| start_hour_13 | 0.776480 |
| start_hour_14 | -3.632531 |
| start_hour_15 | -1.086744 |
| start_hour_16 | 2.667484 |
| start_hour_17 | 2.434955 |
| start_hour_18 | 4.961205 |
| start_hour_19 | -1.084956 |
| start_hour_20 | -0.896456 |
| start_hour_21 | -1.223675 |
| start_hour_22 | -1.140175 |
| start_hour_23 | -0.809480 |

Больших по модулю коэффициентов нет, что свидетельствует об отсутствии переобучения модели.




Оценка на кроссвалидации: $0,95801 \pm 0,01572$.

Обученный классификатор был применён к проверочной выборке и результаты отправлены в соревнование. Получена оценка $0,94466$ (установленный составителями baseline $= 0,92784$):




Имя команды в соревновании - «*[YDF & MIPT] Субботин Сергей Александрович*»

# 6 Заключение

В работе продемонстрированы навыки работы с данными, их предобработкой, анализом и извлечением дополнительных признаков для этих данных; применение алгоритмов машинного обучения, построения моделей и их валидации.

Идеи для дальнейшего улучшения модели:

- проверить признаки, связанные с датой начала сессии на данных без ошибок;
- проверить другие схемы разбиения на сессии;
- проверить композиции алгоритмов;
- добавить предобработку для сайтов — для некторых сайтов существуют несколько имён (с и без префикса *www*, или разные load-balance ноды у некоторых хостингов — например у photobucket.com и др.);
- добавить признаки основанные на внешней информации — например при помощи категоризации сайтов добавить признаки наличия в сессии сайтов относящихся к новостным. или игровым и т.п.

Интересно так же поработать с многоклассовой классификацией. В рамках проекта обучалась модель для многоклассовой классификации на выборке `vw`. Обучалось три классификатора — стохастический градиентный спуск с логистической функцией потерь, логистическая регрессия и классификатор Vowpal Wabbit. Метрикой качества выступает точность (accuracy). Производительность классификаторов приведена в таблице:

| Классификатор | точность на обучающей выборке | точность на отложенной выборке |
|---|---|---|
| SGDClassifier(log) | 0,29365 | 0,18316 |
| LogisticRegression(lbfgs) | 0,36292 | 0,19887 |
| VowpalWabbit | 0,34542 | 0,18768 |

При этом использовалась схема разбиения на сессии 10-10, гиперпараметры не настривались, дополнительных признаков не использовалось.