

Предлагается решить 1 из двух вариантов ДЗ на выбор.

Вариант 1

Введение

Предлагается решить 6 задач используя SparkStreaming и Kafka API.

- ❑ Репозиторий для сдачи: <http://gitlab.atp-fivt.org/hobod2026/.../XXX-rtkafka>
- ❑ Ветки: **rtkafkatask1** - **rtkafkatask6**. В ветках 3 - 6 **тестов нет** поэтому просто залейте вывод и логи в тот же merge request, в которой код. На падения системы тестирования не обращайте внимания.

Задания

Задачи 1,2 решаются с помощью SparkStreaming, задачи 3-6 с помощью Kafka (или связки Kafka со SparkStreaming).

1. [0,5 балла]. Эвристическая сегментация пользователей

Сегмент - это множество пользователей, определяющееся неким признаком.

Когда пользователь посещает web-сервис со своего устройства, это событие логируется на стороне web-сервиса в следующем формате: **user_id <tab> user_agent**.

Например:

```
f78366c2cbed009e1febcb060b832dbe4   Mozilla/5.0 (Linux; Android 4.4.2; T1-701u
Build/HuaweiMediaPad) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.73
Safari/537.36
62af689829bd5def3d4ca35b10127bc5   Mozilla/5.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36
```

На вход поступают порции web-логов в описанном формате. Требуется разбить аудиторию (пользователей) в этих логах на следующие сегменты:

- 1) Пользователи, которые работают в интернете из-под iPhone.
- 2) Пользователи, кот. используют Firefox браузер.
- 3) Пользователи, кот. используют Windows.

Не стоит волноваться если какие-то пользователи не попадут ни в 1 из указанных сегментов поскольку в реальной жизни часто попадают данные, которые сложно классифицировать. Таких пользователей просто не включаем в выборку.

Также сегменты могут пересекаться (ведь возможен вариант, что пользователь использует Windows, на котором стоит Firefox). Для того, чтоб выделить сегменты можно использовать следующие эвристики (или придумать свои):

Сегмент	Эвристика
<code>seg_iphone</code>	<code>parsed_ua['device']['family'] like '%iPhone%'</code>
<code>seg_firefox</code>	<code>parsed_ua['user_agent']['family'] like '%Firefox%'</code>

<code>seg_windows</code>	<code>parsed_ua['os']['family'] like '%Windows%'</code>
--------------------------	---

Оцените кол-во уникальных пользователей в каждом сегменте используя алгоритм [HyperLogLog](#) (поставьте `error_rate` равным 1%).

В результате выведите сегменты и количества пользователей в следующем формате: `segment_name <space> count`. Отсортируйте результат по количеству пользователей в порядке убывания.

Входные данные: `/data/course4/uid_ua_100k_splitted_by_5k`

Пример

<code>...</code>
<code>seg_firefox 4176</code>
<code>...</code>

2. [0,4 балла]. Анализ социального графа

Для каждого пользователя, имеющего идентификатор в колонке `user`, подсчитайте количество общих друзей с другими пользователями, имеющими идентификатор в колонке `user`. Считаем, что 2 пользователя, имеющих идентификатор в колонке `user`, имеют общих друзей, если у них в поле `friends` есть пересечения.

Датасет подаётся на вход частями каждые 10 секунд. **Считаем что пользователи из разных батчей не имеют пересечений по друзьям.**

Когда подача данных закончится, выведите 50 пар пользователей, имеющих наибольшее количество общих друзей. Пары должны быть отсортированы в порядке убывания сначала по количеству общих друзей, затем по 1-му `userId`, и наконец по 2-му. При выводе ответа надо фильтровать одинаковые пары, отличающиеся порядком, так чтобы 1-ый `userId` < 2-го `userId`.

Исходные данные:

- Полный датасет (для сабмита в GitLab): `/data/graphDFQuarter`
- Частичный семпл: `/data/graphDFSsampleQuarter`

Структура данных:

- `userId` - id пользователя
- `friends` - id его друзей (*не все* ID из `friends` имеют свои строки в датасете)

Формат вывода:

`common_friends userId1 userId2`

Часть вывода на данных `graphDFSsampleQuarter`.

<code>...</code>		
<code>1346</code>	<code>10541383</code>	<code>51640390</code>
<code>1334</code>	<code>3812683</code>	<code>38274136</code>
<code>1301</code>	<code>1049214</code>	<code>4685334</code>
<code>1247</code>	<code>49405386</code>	<code>56739429</code>
<code>1197</code>	<code>24170958</code>	<code>24248832</code>
<code>1186</code>	<code>23936386</code>	<code>56739429</code>

```
1180 2408699 33766998
1164 16666475 52992701
...
```

3. [0,2 балла]. Исследование топика

Используя Kafka CLI или Kafka Python API, посчитать кол-во сообщений в топике. Программа должна принимать на вход название топика и выводить кол-во сообщений в нём.

Добавляем Kafka-логику в задачу на [эвристическую сегментацию пользователей](#).

4. [0,1 балла]. Spark Streaming + Kafka, задание 1

Используя [Kafka producer API](#)¹ записать входные данные (из задачи 1) в topic. Каждые 10 секунд должен приходиться новый батч.

5. [0,1 балла]. Spark Streaming + Kafka, задание 2

Доработать код предыдущей задачи так, чтоб она с помощью [KafkaUtils](#) подключалась к topic'у.

6. [0,2 балла]. Spark Streaming + Kafka, задание 3

Добавить в код параметр - номер партиции. Если указан - получаем данные только с неё. Если не указан - также, как и в п.2.

Добавить 2 параметра: начальный Offset и конечный. Если указаны - читаем данные только в указанных Offset'ах.

- Если указан только конечный, то читаем сначала и до указанного Offset'а.
- Если только начальный, то от указанного Offset'а до текущего времени.

Дополнительная информация

1) Код для генерации порций.

Используйте его в **1й задаче** без изменений (можно изменять только DATA_PATH) т.к. он критичен для работы тестирующей системы.

```
from hdfs import Config
import subprocess

client = Config().get_client()
nn_address = subprocess.check_output('hdfs getconf -confKey dfs.namenode.http-address',
shell=True).strip().decode("utf-8")

sc = SparkContext(master='yarn-client')

# Preparing base RDD with the input data
DATA_PATH = "/data/course4/wiki/en_articles_batches"
```

¹ Можно использовать и другую библиотеку, см. здесь <https://www.bigdataschool.ru/blog/kafka-python-clients.html>. Confluent-kafka на данный момент не стоит на кластере, но 2 остальных есть.

```
batches = [sc.textFile(os.path.join(*[nn_address, DATA_PATH, path])) for path in
client.list(DATA_PATH)[:30]]

# Creating QueueStream to emulate realtime data generating
BATCH_TIMEOUT = 2 # Timeout between batch generation
ssc = StreamingContext(sc, BATCH_TIMEOUT)

dstream = ssc.queueStream(rdds=batches)
```

2) Docker-контейнер

В случае проблем с работой кластера, для отладки задач можно воспользоваться данным [Docker-контейнером](#).

Вариант 2

Введение

Предлагается решить 3 задачи используя KafkaStreams API.

- ❑ Репозиторий для сдачи: <http://gitlab.atp-fivt.org/hobod2025/.../XXX-rtkafka>
- ❑ Ветка: **rtkafkatask7** (1 ветка для всех задач).

Задание построено по мотивам примеров, рассмотренных на конференции (видеозаписи: [1-я лекция](#) и [2-я лекция](#)).

Необходимо скопировать (`git pull`) [данный репозиторий](#)² затем залить его в **rtkafkatask7**. Заливать нужно весь проект, в том числе `.gitlab-ci.yml`. На `run.sh`, который уже лежит в репозитории, не обращайте внимания.

Далее нужно дополнить код в модуле `homework`:

1. Создать топологию, которая имеет следующие три выходных топика:
 - **[0,4 балла]** таблица, ключом которой является имя игрока, а значением -- сумма ставок, произведённых игроком. Записывается в топик **bettor-amounts**
 - **[0,4 балла]**. таблица, ключом которой является имя команды, а значением -- сумма ставок на эту команду (ставки на "ничью" в подсчёте игнорируются). Записывается в топик **team-amounts**
 - **[0,7 баллов]**. поток, ключом которого является имя игрока, а значениями -- подозрительные ставки. Подозрительными считаем ставки, произведённые в пользу команды в пределах одной секунды до забития этой командой гола. Записывается в топик **possible-frauds**
2. С использованием `TopologyTestDriver` в модуле `homework` уже написаны модульные тесты, но все они помечены как `@Disabled` (иначе они упадут, т. к. реализация топологии отсутствует). Перед началом работы необходимо убрать аннотации `@Disabled` и в процессе выполнения работы необходимо добиться того, чтобы тесты выполнялись. Также в этом модуле настроены правила `Checkstyle`.

² За основу берите именно этот репозиторий, а не оригинальный репозиторий Ивана Пономарёва т.к. в данном репозитории есть GitLab CI, а в исходном - есть Github Actions, который не будет работать в GitLab.

3. При проверке задания будут учитываться:

- Успешное прохождение сборки со стандартными тестами и правилами Checkstyle является минимально необходимым условием для приёма домашнего задания.
- работоспособность программы как сервиса: программа должна запускаться, присоединяться к кластеру Kafka и обрабатывать данные, подаваемые программой в модуле Producer
- Наличие и качество дополнительных тестов приветствуется.
- эффективность созданной топологии, отсутствие излишних репартицирований