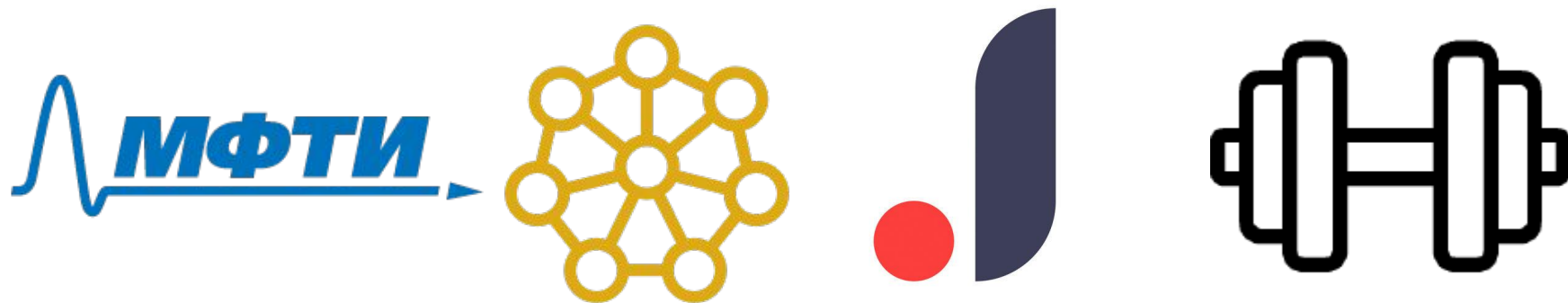


# “Публичные решения и щепотка соли”






Обзор 3-го решения задачи прогнозирования  
следующей покупки

Михаил Трофимов

Обо мне



# Почему я сегодня здесь?

#	Команда	
1	aprotopopov	
2	ssh1	
3	вжух-вжух и в продакшн	
4	avolchek	
5	vadimfb	

## retailhero-recomender-baseline

Бэйслайн к задаче RetailHero.ai/#2 от @geffy 💪

data-science

recommender-system

contest-solution

Python MIT 13 70 0 0 Updated 27 days ago

# Постановка задачи



## Какие товары клиент купит в следующий раз?

Сделай сервис, который сможет выдержать нагрузку и будет угадывать следующую покупательскую корзину.

Участвовать в соревновании

# Постановка задачи

## Задача

Строим полноценную рекомендательную систему! Участникам необходимо разработать сервис, который сможет отвечать на запросы с предсказаниями будущих покупок клиента и при этом держать высокую нагрузку. По информации о клиенте и его истории покупок необходимо построить ранжированный список товаров, которые клиент наиболее вероятно купит в следующей покупке.

## Критерий качества

Решение должно представлять собой архив с кодом сервиса. Метрика качества - **MNAP@30 (Mean Average Precision)**, средний по всем запросам нормализованный Average Precision. Решение должно обязательно уложиться в ограничения по ресурсам. Если решение не выдерживает нагрузки или отвечает на запросы слишком долго — оно не будет считаться успешно прошедшим тестирование. Запросы поступают в строго хронологическом порядке.

# Постановка задачи

## Задача

Строим полноценную рекомендательную систему! Участникам необходимо разработать сервис, который сможет отвечать на запросы с предсказаниями будущих покупок клиента и при этом **держат высокую нагрузку**. По информации о клиенте и **его истории покупок**

**Не самая стандартная задача, предпросчет по пользователям выпадает**

## Критерий качества

**Решение должно представлять собой архив с кодом сервиса.** Метрика качества -

**Сдаем код, не увидим тестовые данные**

средний по всем запросам нормализованный Average Precision.

Решение должно обязательно уложиться в

**ограничения по ресурсам. 1 GB на данные, выдержит 5 секунд на инициализацию**

запросы слишком долго — оно не будет

считаться успешно прошедшим

тестирование. **Запросы поступают в строго хронологическом порядке.**

**Лика во время тестирования не будет :(**

# Решение как инфраструктура

- препроцессинг
- разбиение данных
- валидационный цикл
- подготовка сабмита в правильном формате
- конвертация данных в спарс-формат
- переиспользуемые утилиты

Все эти компоненты присутствуют в любом решении, но не являются решением. *Делиться\** ими полезно и безопасно.

*\*призываю вас так и делать*

# Препроцессинг

purchases.csv

client_1	transaction_1
client_1	transaction_2
client_1	transaction_3
client_2	transaction_4
...	...
client_M	transaction_N

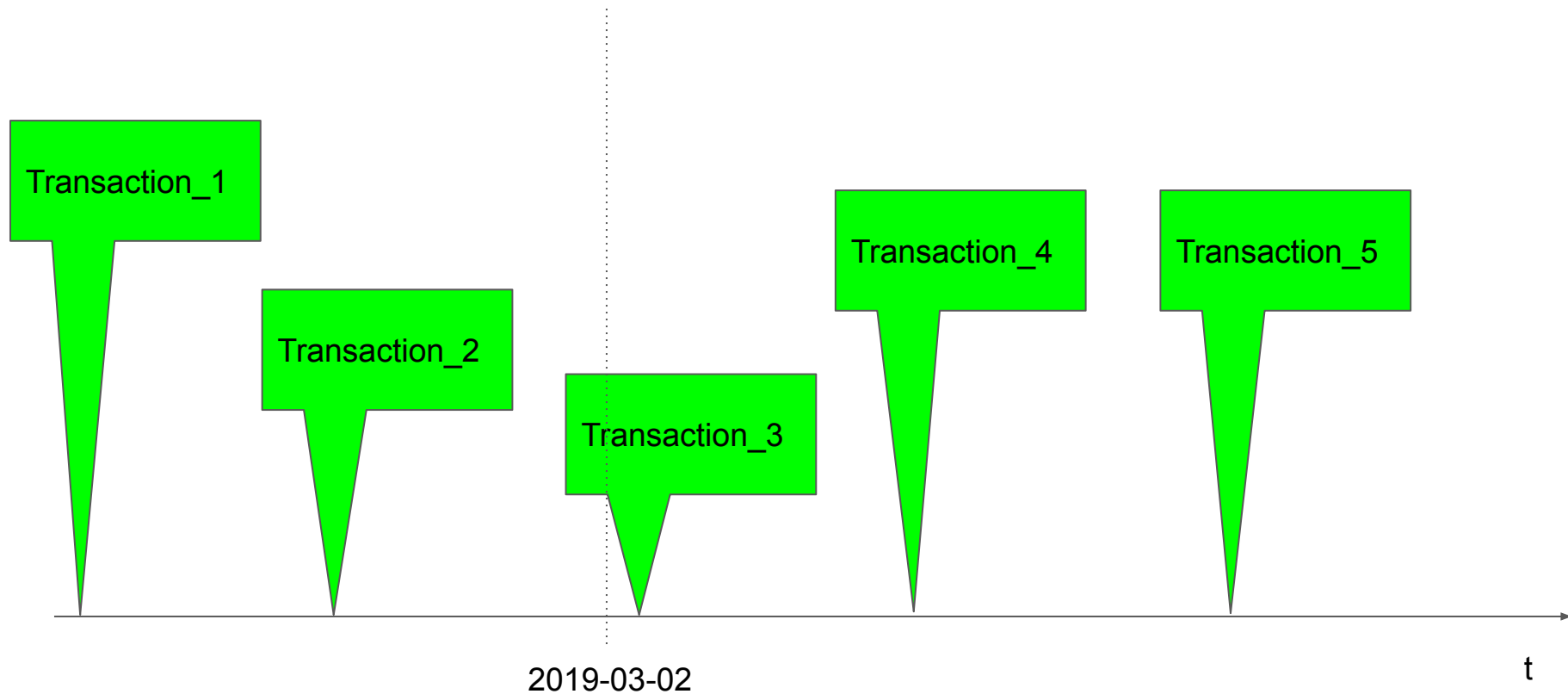
16 шардов

01.jsons

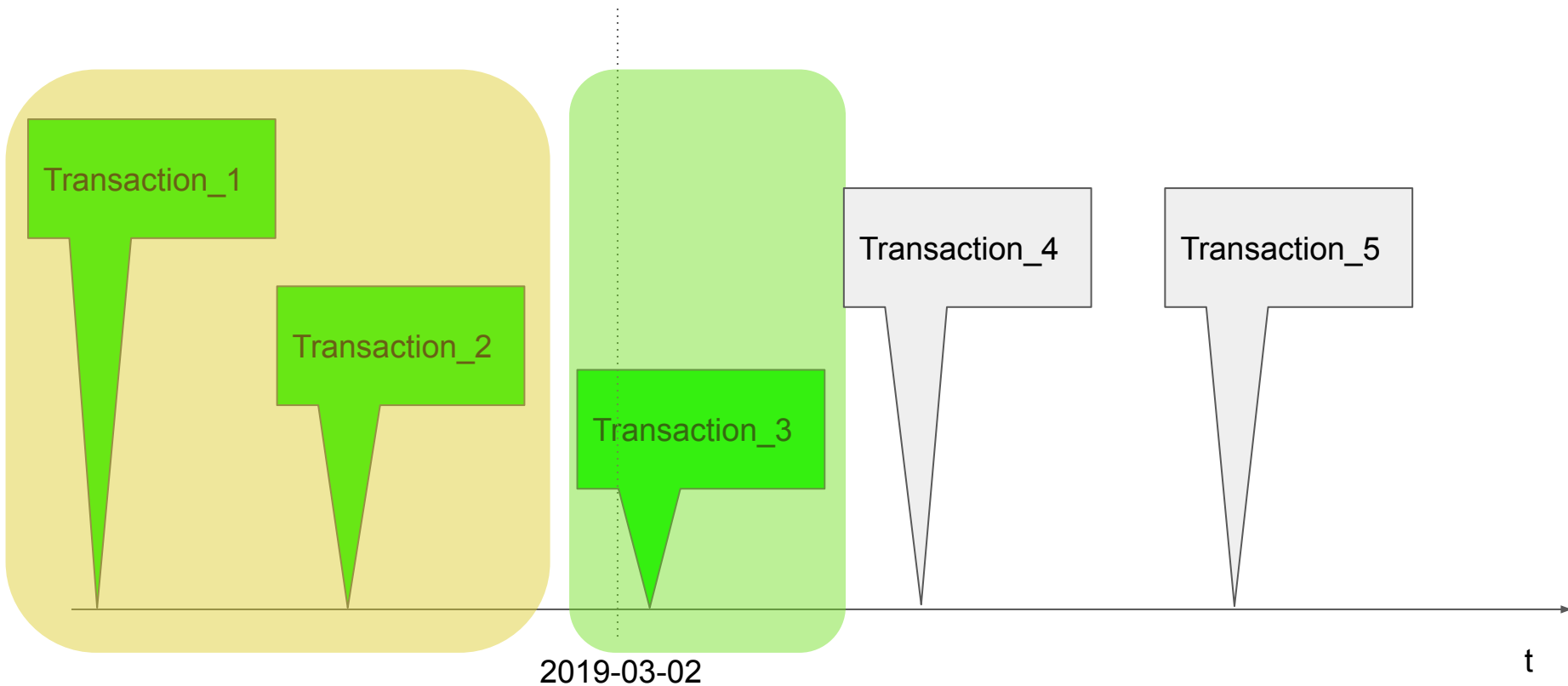
client_1	transaction_{1, 2, 3}
client_17	transaction_{X, Y, X}
...	...



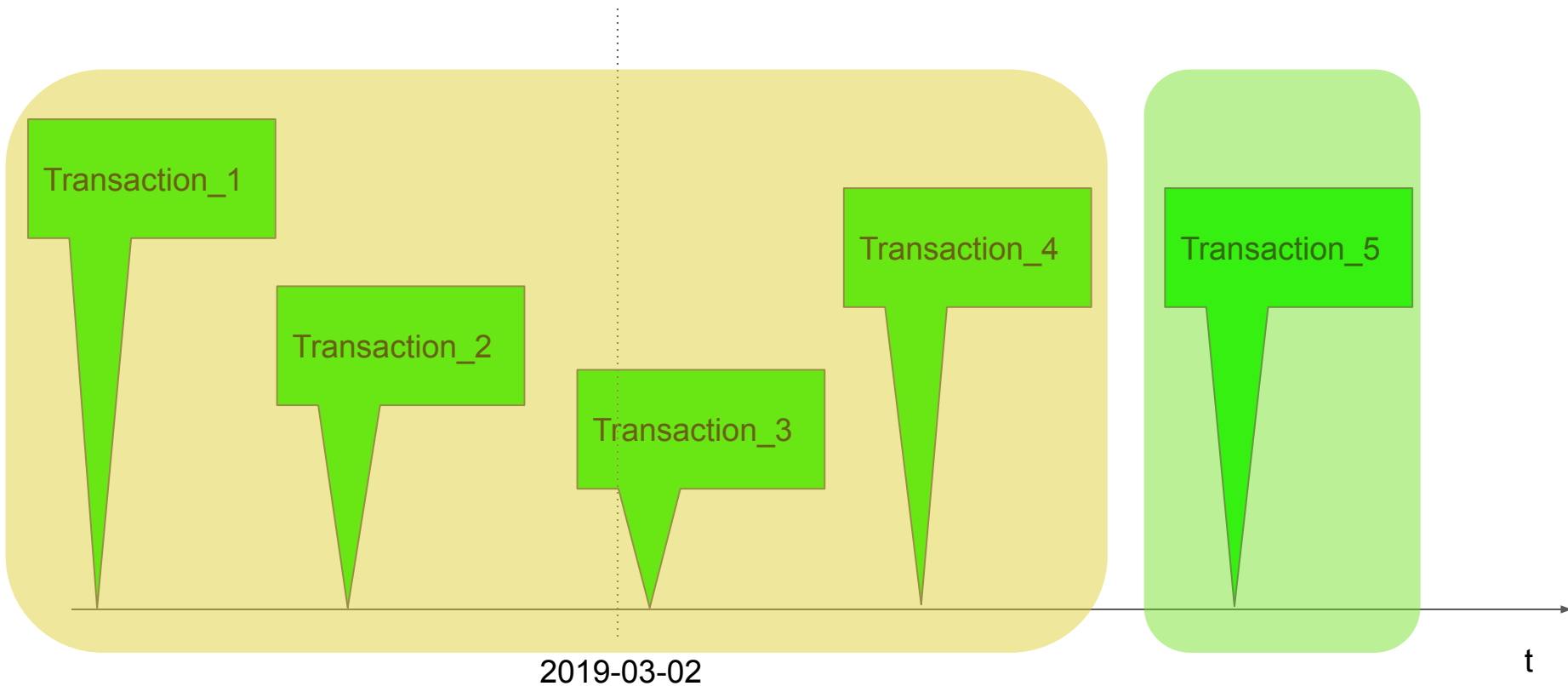
# Валидация



# Валидация



# Валидация



Какой из двух представленных вариантов валидации  
лучше?

Какой из двух представленных вариантов валидации лучше?

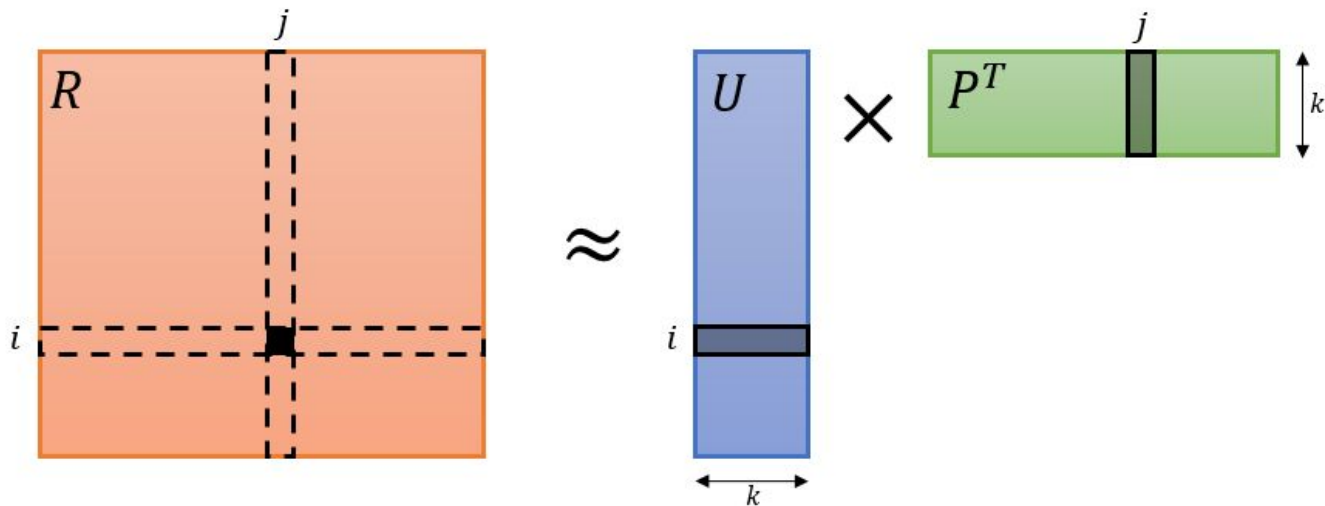
Второй

считаться успешно прошедшим

тестирование. Запросы поступают в строго  
хронологическом порядке.

+ экспериментальная проверка

# Модели: implicit.ALS



- Реализация в `implicit` позволяет на лету пересчитывать профиль пользователя по истории
- Я так и не смог успешно завести эту модель

# Модель: nn\_v1

- (a.k.a. кастомная матричная факторизация с триплет лоссом)
- (a.k.a. кастомная BPR матричная факторизация)
- (a.k.a. кастомные эмбединги)

```
class UserModel(nn.Module):  
    def __init__(self, num_products, embedding_dim):  
        super(UserModel, self).__init__()  
        self._model = nn.Linear(num_products, embedding_dim)  
  
class ItemModel(nn.Module):  
    def __init__(self, num_products, embedding_dim):  
        super(ItemModel, self).__init__()  
        self._embeds = nn.Embedding(num_products, embedding_dim)
```

# Быстрый поиск соседей: FAISS

```
# export knn index (compression and speed-up by FAISS, with Inner Product as distance)
import faiss

quantizer = faiss.IndexFlatIP(dim)
index = faiss.IndexIVFPQ(quantizer, dim, 128, 16, 8)
index.train(item_vectors)
index.add(item_vectors)
faiss.write_index(index, output_dir + "/knn.idx")
```

nn\_v1 + faiss не дали ожидаемого прироста, но задел был классный :)



# Модель: item2item

 [artyerokhin](#) / [x5-retailhero-implicit-baseline](#)

```
# Initialize model  
model = implicit.nearest_neighbours.TFIDFRecommender(K=50)
```

# Модель: item2item

 [artyerokhin](#) / [x5-retailhero-implicit-baseline](#)

```
# Initialize model  
model = implicit.nearest_neighbours.TFIDFRecommender(K=50)
```

# Модель



artyer

```
# Initial
```

```
model =
```

```
(k=50)
```



# Model



artyer

```
# Initial  
model =
```

## Contents:

- Quickstart
  - Installation
  - Basic Usage
  - Articles about Implicit
  - Requirements
- RecommenderBase
- Alternating Least Squares
- Bayesian Personalized Ranking
- Logistic Matrix Factorization
- Approximate Alternating Least Squares
  - NMSLibAlternatingLeastSquares
  - AnnoyAlternatingLeastSquares
  - FaissAlternatingLeastSquares

(k=50)

# Итого (LB: 0.1137)

Что было опубликовано:

- инфраструктурные запчасти
- ALS
- item2item
- кастомные эмбединги на нейросети
- user2user (добавилось позже)

Что не было:

- переранжирование бустингом
- Replay-baseline\*

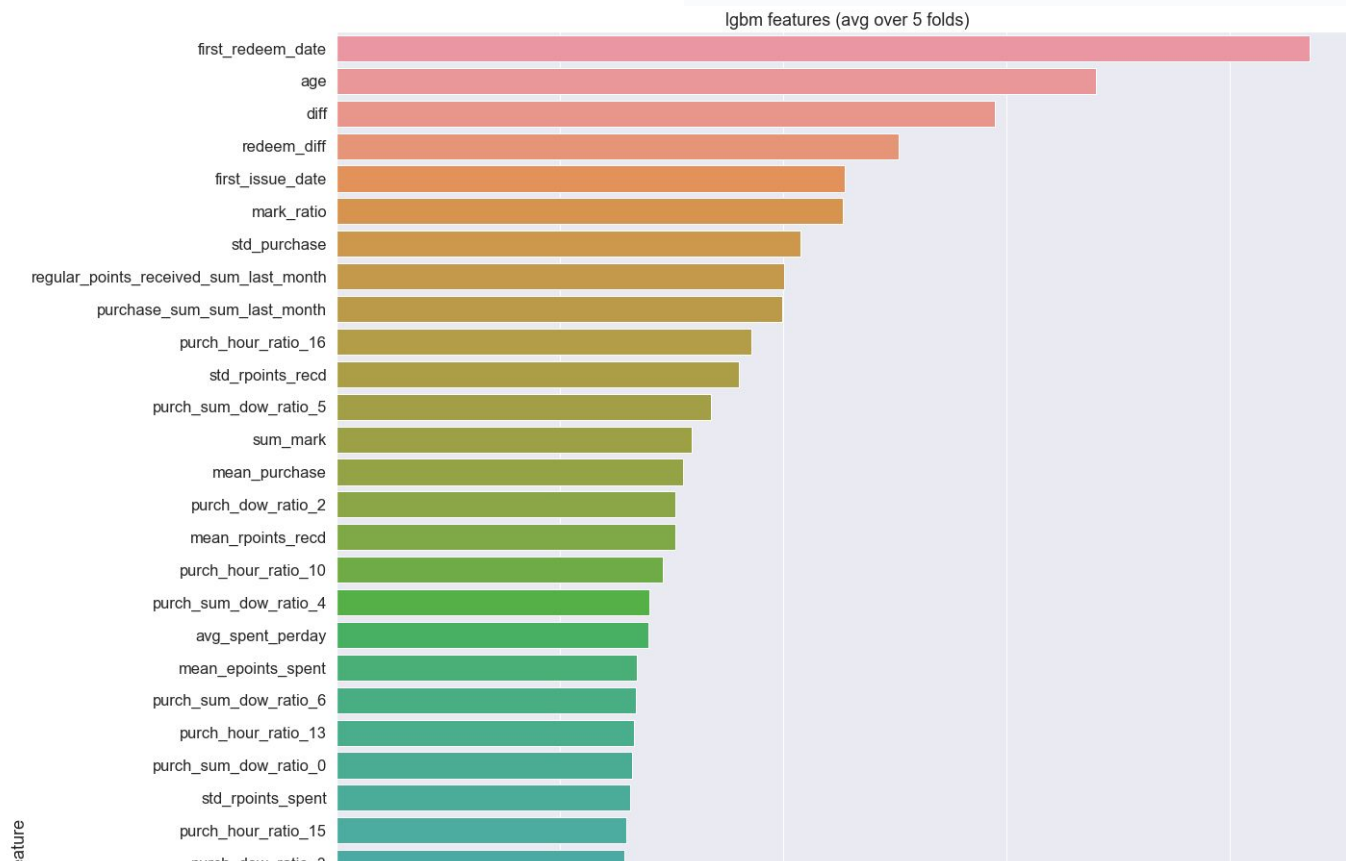
# Дальнейшие шаги (LB: 0.1272)

- Увеличение покрытия за счет разных i2i-моделей: cos1, tfidf1, tfidf10
- i2i-модель по категориям L4
- Был ли продукт в истории клиента
- Стандартные фичи:
  - частоты по категориям
  - средняя цена покупаемых товаров
  - текущий день недели
  - средняя масса покупаемых продуктов
  - длина истории
  - ...
- Отбор 150 кандидатов по i2i-tfidf30 модели (recall: ~0.44)
- CatBoost (loss=QuerySoftMax) в качестве переранжирующей модели

# Вдохновение для фичей:



olegitor / X5.Uplift.public



# Модель: nn\_v2

- (a.k.a. MLP + BCELoss)
- (a.k.a. упрощать дальше некуда)

```
class AwesomeModel2(nn.Module):  
    def __init__(self, num_products):  
        super(AwesomeModel2, self).__init__()  
        self._fc1 = nn.Linear(num_products, 512)  
        self._fc2 = nn.Linear(512, num_products)
```

Добавил как фактор в миксер: LB 0.1272 -> 0.1292



# Актуальность айтемов

 artyerokhin / x5-retailhero-implicit-baseline

```
In [5]: contemporary_items = df_purchases[df_purchases["transaction_datetime"] >
                                             "2019-02-15 00:00:00"]["product_id"].unique()
```

```
In [6]: # небольшой хак – давайте будем рекомендовать только "современное"
df_purchases = df_purchases[df_purchases["product_id"].isin(contemporary_items)]
```



# Актуальность айтемов

- Переобучил i2i-модели на “актуальных” айтемах
- Аналогично для nn\_v2
- Добавил product\_first\_seen\_date, product\_last\_seen\_date как фичи

**LB: 0.1292 -> 0.1306**

## 2-этапное реранжирование

- Собираем кандидатов (~400 в среднем получалось):
  - история покупок
  - топ-50
  - i2i модели
- Учим на i2i и nn\_v2 фичах ранжирующую формулу
  - CatBoost(loss=PairLogit)
- Отбираем 250 кандидатов
  - Recall@250: 0.47 -> 0.54

**LB: 0.1306 -> 0.1311**

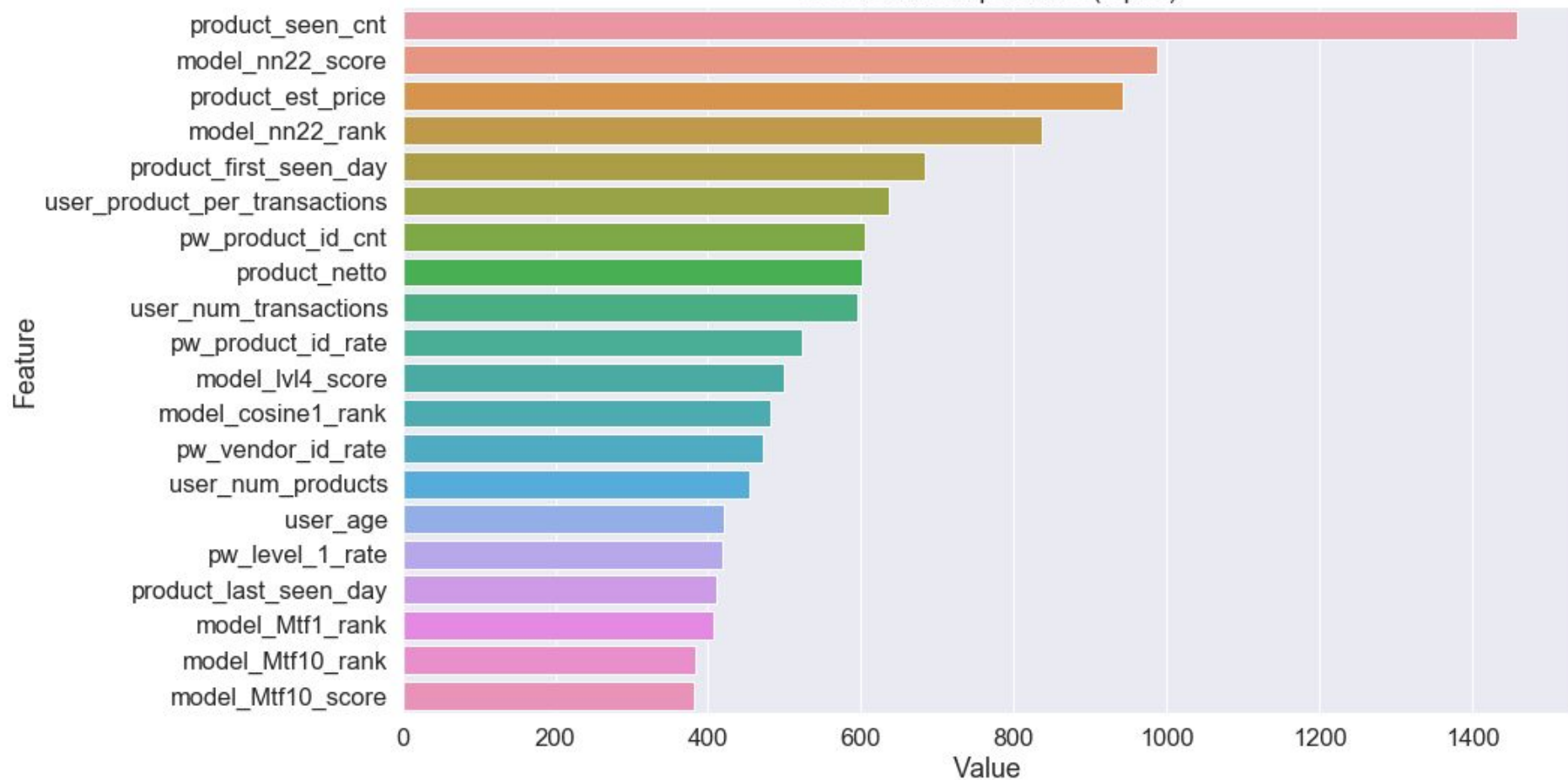
# Финальный блендинг

- Переранжируем 250 кандидатов с помощью 2 моделей
  - CatBoost(loss=QuerySoftMax)
  - LightGBM(target=lambdarank)
- Усредняем скоры 1:1

**LB: 0.1311 -> 0.1330**

**Лучшим на привате оказалось “консервативное” решение**

LGB feature importance (top20)



# Что не взлетело

- user-2-user
- ALS
- FM
- LSTM/GRU
- BPR-like лоссы
- Дневные топы из будущего
- Использование магазинов в любом виде
- CatBoost(langevin)
- LightGBM(target=rank\_xendcg)
- SVD от пользовательской истории
- TF-IDF/Hashing над пользовательской историей

Спасибо за внимание!

PS: код я когда-нибудь выложу

ods: @mtrofimov

mail: [mikhail.trofimov@phystech.edu](mailto:mikhail.trofimov@phystech.edu)