

**КАКИЕ ТОВАРЫ
КЛИЕНТ КУПИТ В
СЛЕДУЮЩИЙ РАЗ?**



Протопопов Антон

Data Scientist, Aarki

✉ : anton@aarki.com

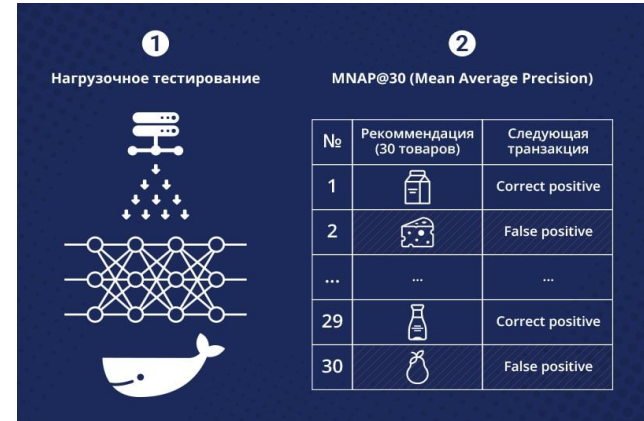
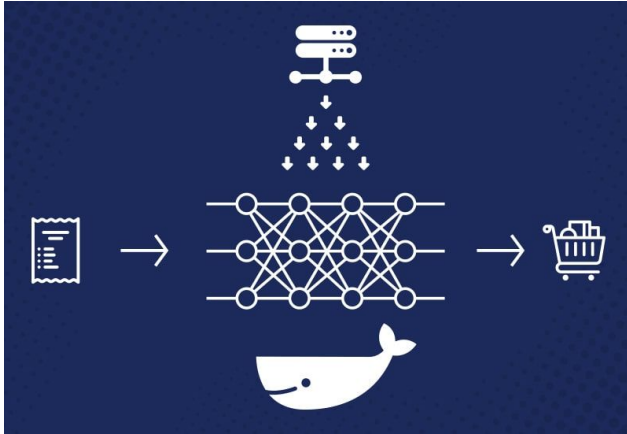
🤖 : @aprotopopov

План доклада

- О конкурсе
- Решение
 - Подготовка данных
 - Отбор кандидатов
 - Ранжирование
- Решения других участников
- Заключение

Постановка задачи

- Предсказать покупки пользователя на основании его транзакций
- Клиенты в тесте не пересекаются с трейном
- Контейнерное соревнование, решение - код алгоритма в zip архиве



Данные

- `clients.csv` - 400k
- `products.csv` - 43k
- `purchases.csv` - 46M

Валидация/тест, известно что даты запроса > 2019-03-01

- `check` - 100 пользователей
- `public` ~ 1k
- `private` ~ 10k

Технические требования

- 8 Гб оперативной памяти, 4 vCPU
- время на подготовку к работе: 5 секунд
- 20 RPS (запросов в секунду)
- ограничение по времени запроса:
 - максимальное 1 сек
 - 95% запросов должны укладываться в 0.3 сек
- максимальный размер упакованного и распакованного архива с решением: 1 Гб

Метрика, MNAP@30

$$MNAP = \frac{1}{|Q|} \sum_{q \in Q} \frac{AP(q)}{\text{IdealAP}(q)}$$

$$AP(q) = \frac{1}{30} \sum_{k=1}^{30} \text{Precision@}k(q)$$

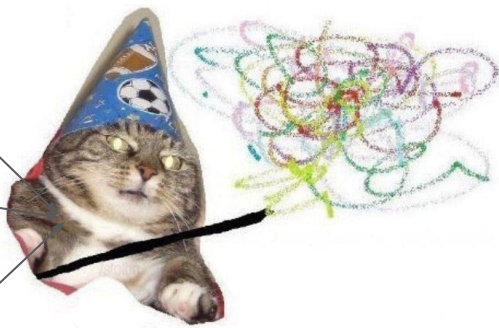
- Q - множество запросов
- $\text{Precision@}k(q)$ - доля релевантных товаров в первых k позициях списка рекомендаций
- $\text{IdealAP}(q)$ - максимальное значение $AP(q)$

Подготовка данных

clients.csv

products.csv

purchases.csv

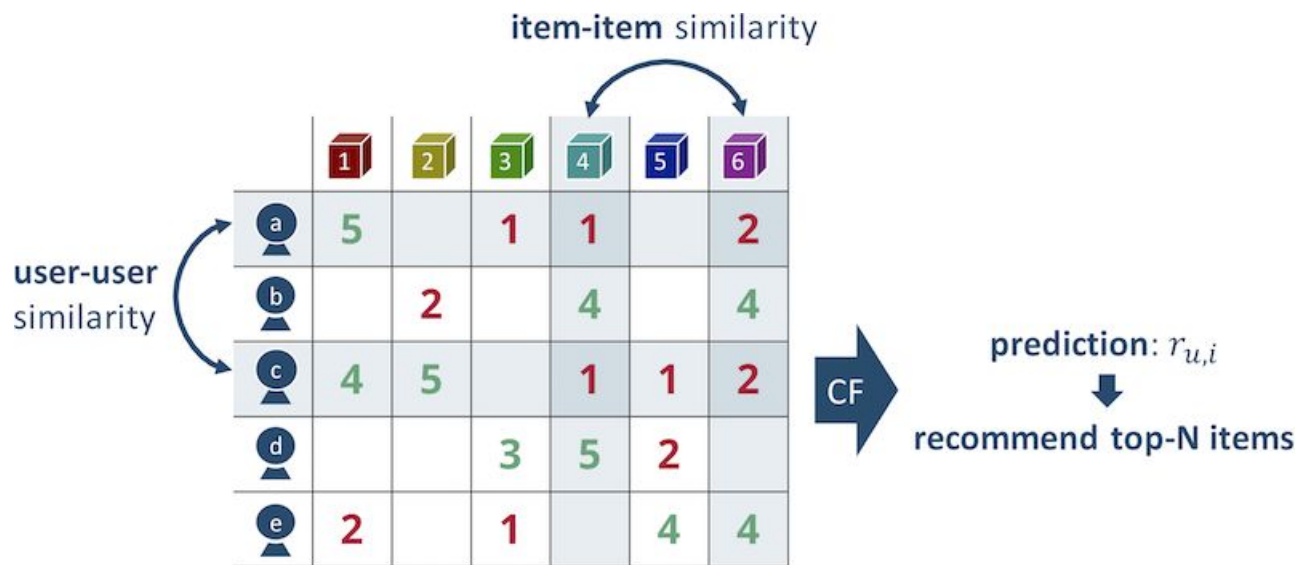


<https://github.com/datagym-ru/retailhero-recomender-baseline>

```
{  
  "query_time": "2019-03-01T05:59:27",  
  "client_id": "001e840150",  
  "age": 71,  
  "gender": "U",  
  "transaction_history": [  
    {  
      "datetime": "2019-01-15T11:53:01",  
      "store_id": "bc09d51b34",  
      "purchase_sum": 308.0,  
      "products": [  
        {  
          "product_id": "dc2001d036",  
          "quantity": 1  
        },  
        ...  
      ]  
    },  
    ...  
  ]  
}
```

Отбор кандидатов

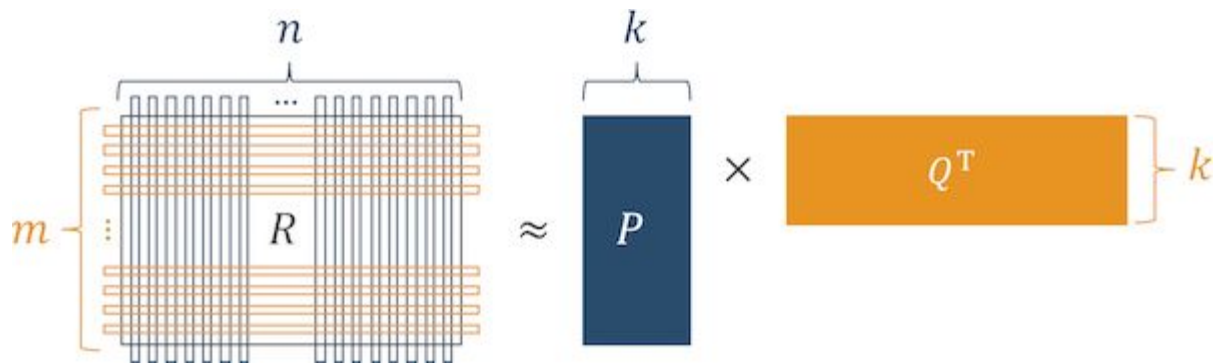
Коллаборативная фильтрация, k-Nearest Neighbors



https://takuti.github.io/Recommendation.jl/latest/collaborative_filtering/

Отбор кандидатов

Матричная факторизация



https://takuti.github.io/Recommendation.jl/latest/collaborative_filtering/

Модели и эвристики для подбора кандидатов

- item2item модели
 - `implicit.nearest_neighbours.CosineRecommender` и `TFIDFRecommender`
- Факторизационная машина
 - `implicit.als.AlternatingLeastSquares`
- топ покупаемые товары
- история покупок пользователя
- user2user модель
 - сжатие `TruncatedSVD` для получения dense матрицы из матрицы покупок пользователей
 - построение индекса FAISS
 - поиск ближайших соседей с FAISS

Итоговый вариант отбора кандидатов

✓ i2i tf-idf100
✓ i2i cosine50
✓ i2i cosine2
✓ global_top@N
✓ user_history@0.5N

✗ ALS
✗ u2u FAISS

| N_pool | mean_N | min_N | max_N | recall |
|--------|------------|-------|-------|----------|
| 30 | 70.760409 | 30 | 110 | 0.399408 |
| 50 | 101.300616 | 50 | 153 | 0.432926 |
| 75 | 144.910696 | 75 | 230 | 0.467893 |
| 100 | 191.008328 | 100 | 271 | 0.497270 |
| 125 | 236.923014 | 125 | 331 | 0.517472 |
| 200 | 375.113246 | 200 | 520 | 0.561788 |
| 500 | 962.501518 | 500 | 1175 | 0.655344 |

Ранжирование

Градиентный бустинг

Microsoft
LightGBM

dmlc
XGBoost

 CatBoost



Градиентный бустинг

Microsoft
LightGBM

1. `binary`
2. `lambdarank`
3. `rank_xendcg`

dmlc
XGBoost

1. `binary:logistic`
2. `rank:pairwise`
3. `rank:ndcg`
4. `rank:map`



 **CatBoost**

1. `RMSE`
2. `QueryRMSE`
3. `PairLogit`
4. `PairLogitPairwise`
5. `YetiRank`
6. `YetiRankPairwise`

Градиентный бустинг

Microsoft
LightGBM

- ✓ binary
- ✓ lambdarank
- ✗ rank_xendcg

dmlc
XGBoost

1. binary:logistic
2. rank:pairwise
3. rank:ndcg
4. rank:map



CatBoost

- ✓ RMSE
- ✗ QueryRMSE
- ✗ PairLogit
- ✗ PairLogitPairwise
- ✗ YetiRank
- ✗ YetiRankPairwise

Признаки, основанные на со-встречаемости товаров

Идея из [описания решения RecSys 2018](#) командой из Авито

$n_{i,j}$ - количество транзакций, содержащих товары i и j вместе

n_i - количество транзакций, содержащих товары i

w_i - нормализованный вес товара i из транзакций пользователя

Транзакция t состоит из товаров p_1, p_2, \dots, p_n

Для каждого товара считаем скоры $n_{p,p_1}, \dots, n_{p,p_n}$

Как фичи используются нормализованные значения $\frac{n_{p,p_n}}{n_{p_n}}$ и статистики: минимум, максимум, сумма.

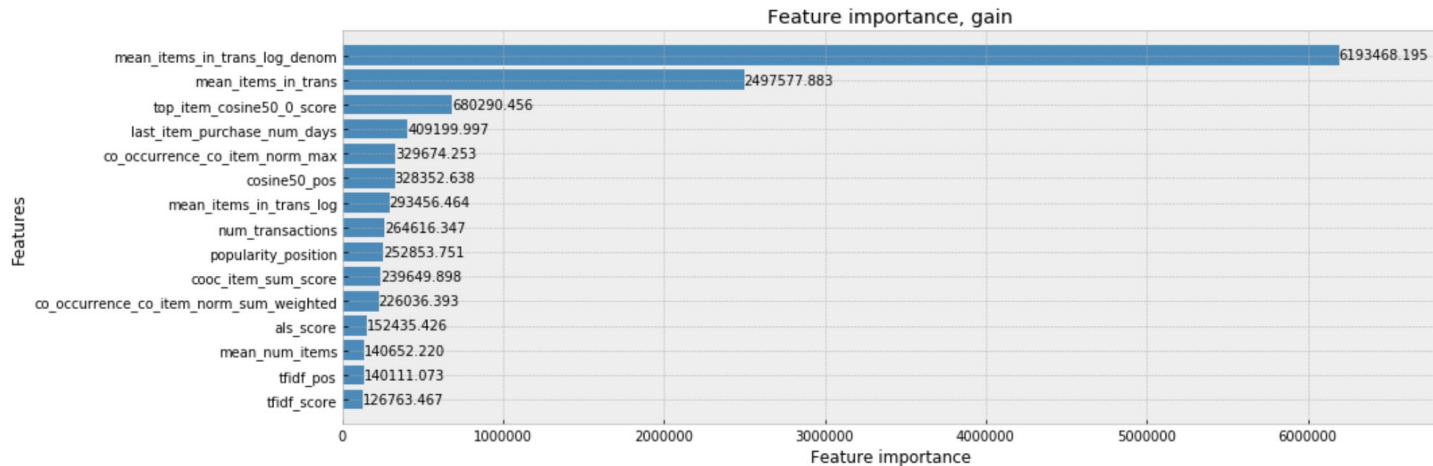
Также эти скоры дополнительно взвешивались на w_i : $w_i \cdot \frac{n_{p,p_n}}{n_{p_n}}$

Другие признаки

- Релевантность и позиция товара от каждой модели
- Среднее количество товаров в транзакции, истории и их производные
- Временные признаки, был выбран товар в последней транзакции, количество дней после последней транзакции
- UMAP эмбединги пользователей на транзакционной активности
- Признаки пользователя из `clients.csv` - возраст, пол
- Признаки товаров из `products.csv` - категориальные признаки

Важность признаков

- Всего ~80 признаков
- Наибольший вклад вносит признаки из истории транзакций пользователей
- Важность топ 15 признаков



Оптимизация скорости

- Кеширование и сохранение в `pickle/feather` признаков по товарам
- Оценка релевантности товаров `i2i` путем ручного перемножения матриц
- Сабсет `sparse` матрицы для каждого пула/истории пользователя, конвертация в `numpy` для быстрого доступа и подсчета фичей

Используемые ресурсы

- Mac Pro, i7
- 32 Gb RAM
- 80 Gb SSD swap
- ~100 Gb disk space



Не сработало

- `word2vec`, обученный на транзакциях пользователей
- `user2user cosine similarity + FAISS`, замедляло работы и не давало сильного буста
- `CatBoost` ранжирующие алгоритмы
- Добавление `ground_truth` товаров и исключение пользователей с отсутствием таргета в пуле товаров
- Получение эмбедингов категорий (`store_id, age`) из обученной `lightfm` модели














Фейлы

- Использование фичей основанных на порядке (количество дней до последней транзакции), хотя в `check` и `public` история транзакция не отсортирована.
- Не хватило 1 часа на тренировку лучшего решения, на локальной валидации скор 0.1558, против финального 0.1550.
- Модель `cosine50` не получилось воспроизвести offline















Результаты соревнования

Public

| # | Команда | Точность | Последнее решение | Попыток |
|----|--|----------|------------------------|---------|
| 1 | vadimfb  | 0,1358 | 23 февраля 2020, 13:19 | 131 |
| 2 | apropopov  | 0,1350 | 23 февраля 2020, 23:51 | 119 |
| 3 | ssh1  | 0,1336 | 23 февраля 2020, 23:50 | 164 |
| 4 | avolchek  | 0,1335 | 23 февраля 2020, 15:31 | 162 |
| 5 | вжух-вжух и в продакшн  | 0,1330 | 23 февраля 2020, 16:50 | 33 |
| 6 | Make me apologize if you can    | 0,1324 | 23 февраля 2020, 22:58 | 60 |
| 7 | greenwolf  | 0,1295 | 23 февраля 2020, 22:24 | 59 |
| 8 | anatoly  | 0,1294 | 23 февраля 2020, 23:06 | 127 |
| 9 | Рекомендации Красная Цена  | 0,1286 | 22 февраля 2020, 22:48 | 96 |
| 10 | antitak   | 0,1283 | 23 февраля 2020, 21:56 | 128 |

Private

| # | Команда | Финальная точность  | Response Time (avg / 95p, ms) | Последнее решение | Попыток |
|----|---|--|-------------------------------|------------------------|---------|
| 1 | apropopov  | 0,148325 | 26,9 / 36,0 | 23 февраля 2020, 23:51 | 119 |
| 2 | ssh1  | 0,147026 | 75,0 / 88,1 | 23 февраля 2020, 23:50 | 164 |
| 3 | вжух-вжух и в продакшн  | 0,145728 | 51,2 / 68,2 | 23 февраля 2020, 16:50 | 33 |
| 4 | avolchek  | 0,145644 | 9,6 / 14,0 | 23 февраля 2020, 15:31 | 162 |
| 5 | vadimfb  | 0,145443 | 20,9 / 23,6 | 23 февраля 2020, 13:19 | 131 |
| 6 | anatoly  | 0,144031 | 71,1 / 122,6 | 23 февраля 2020, 23:06 | 127 |
| 7 | antitak   | 0,143769 | 17,1 / 31,0 | 23 февраля 2020, 21:56 | 128 |
| 8 | greenwolf  | 0,143422 | 20,0 / 28,4 | 23 февраля 2020, 22:24 | 59 |
| 9 | Рекомендации Красная Цена  | 0,143019 | 30,0 / 43,2 | 22 февраля 2020, 22:48 | 96 |
| 10 | Пропустите навстречу, пожалуйста  | 0,141580 | 67,0 / 122,0 | 23 февраля 2020, 23:03 | 31 |

Решение второго места, @ssh1

- В качестве кандидатов вся история клиентов, если меньше 100 продуктов - из глобального топа
- 3 типа фич, `Client`, `Product`, `Client-Product`
- Из эмбедингов - `prod2vec`, 2 фичи, расстояние от продукта до среднего от продуктов клиента и до `max`
- Заказ товара в последних пяти транзакциях в виде последовательности бит (категориальная). Например `10011` - купил этот товар в последней, предпоследней и 5 транзакций назад
- CatBoost с YetiRank, `lr=1`, валидация по `MAP@30`
- Скор на локальном тесте 0.154, на паблице 0.133

Решение шестого места, @anatoly_i

- 1) берём товары, которые клиент уже покупал (до предсказываемой транзакции)
- 2) добавляем 50 самых популярных товаров
- 3) строим на них классификатор `lightgbm` (будет в следующей покупке или нет)
- 4) перебором остались такие фичи:
 - категориальные `segment_id`, `level_2`, `level_3`, `vendor_id`
 - нормированная частота товара для каждого клиента за всё время
 - время, прошедшее с момента последней покупки товара до момента последней известной транзакции клиента + ранги по этой фиче
 - частота товара среди всех клиентов
 - количество магазинов, в которых встречался этот товар
 - количество транзакций клиента
 - стандартное отклонение количества продуктов в транзакциях клиента
 - количество продуктов в последней транзакции клиента / среднее кол-во продуктов в транзакции клиента
- 5) обучался на 2-3 последних транзакциях всех клиентов и складывал вероятности из 6 немного отличающихся моделей

Заключение

- Контейнерные соревнования стимулируют писать основной код в скриптах, а не ноутбуках
- Контейнер позволяет полностью воспроизвести решение, даже если локальный код/файлы уже безвозвратно изменены
- В условиях ограничения на ресурсы приходится взвешивать стоит ли включать в решение дорогие для расчета признаки/подходы
- Градиентный бустинг быстро работает и может использоваться для real-time рекомендаций. Даже на 6000 итераций с общим пулом ~ 200 кандидатов и ~ 80 фичей обрабатывает в заданные рамки.

Спасибо за внимание!

Решение: https://github.com/aprotopopov/retailhero_recommender

✉ : anton@aarki.com

🧐 : @aprotopopov