Рекомендательные системы

ШАД

Весна 2025

Лекция 3:
Кандидаты, модели

О чем сегодня

- устройство типичного рекомендера
- виды кандидатогенераторов
- про ANN-индексы
- коллаборативную фильтрацию
- и по мелочи

Флоу работы типичного рекоммендера

- Запрос прилетает в ваш рантайм
- Быстрыми, но часто сложными эвристиками генерируем кандидатов ✨
- Несколькими ранжирующими моделями сужаем набор айтемов 🔎
  - Например, для пары (user, item) считаем скор, сортируем по нему, оставляем топ
  - Так сначала моделями потоньше, потом – потолще
- Накручиваем бизнес-логику 💰
- Отдаем выдачу
- Пользователь жестко кайфует 🤗

Кандидатогенерация

Зачем это вообще:

- обычно на обработку всего запроса у вас есть порядка 100-1000мс в зависимости от вашего продукта
- если у вас миллиарды айтемов, сделать осмысленное предсказание моделью для пары (user, item) для всех айтемов вы не успеете

Что делаем:

- эвристиками наберем так, чтобы попало в топ по скору вашей модели ранжирования
- Либо просто оптимизируем полноту.

Ранжирование

В базовом варианте строится модель, решающая задачу бинарной классификации:

- **Позитивный класс – целевое взаимодействие: покупка, клик, долгий просмотр**
- **Негативный класс – сэмплинг или показы**

В более продвинутом – используют ранжирующий лосс:

- **LambdaMART**
- **QueryCrossEntropy**
- **YetiRank**

Но про это в следующей лекции

Кандидатогенерация

Иногда не нужна

Если ваше множество айтемов небольшое – можно успеть поскорить все.

Иногда ее недостаточно

Если у вас совершенно монструозное множество айтемов, его часто полезно шардировать – разложить ваше множество айтемов непересекающимся образом на разные тачки и параллельно набирать на них по отдельности (от shard = осколок)

Но это стоит денег.

Самый полезный слайд во всем курсе

(еще один)

- кандидатов часто нужно фильтровать
- например, по наличию товара на складе или по доступности сериала в регионе
- критически важно фильтровать кандидатов на ранних стадиях
- а на поздних – не фильтровать
- чтобы не тратить ценный ресурс ранжирующей модели на кандидатов, которых вы потом все равно фильтранете
- и не плодить короткие выдачи

Какие бывают кандгены

Статические топы

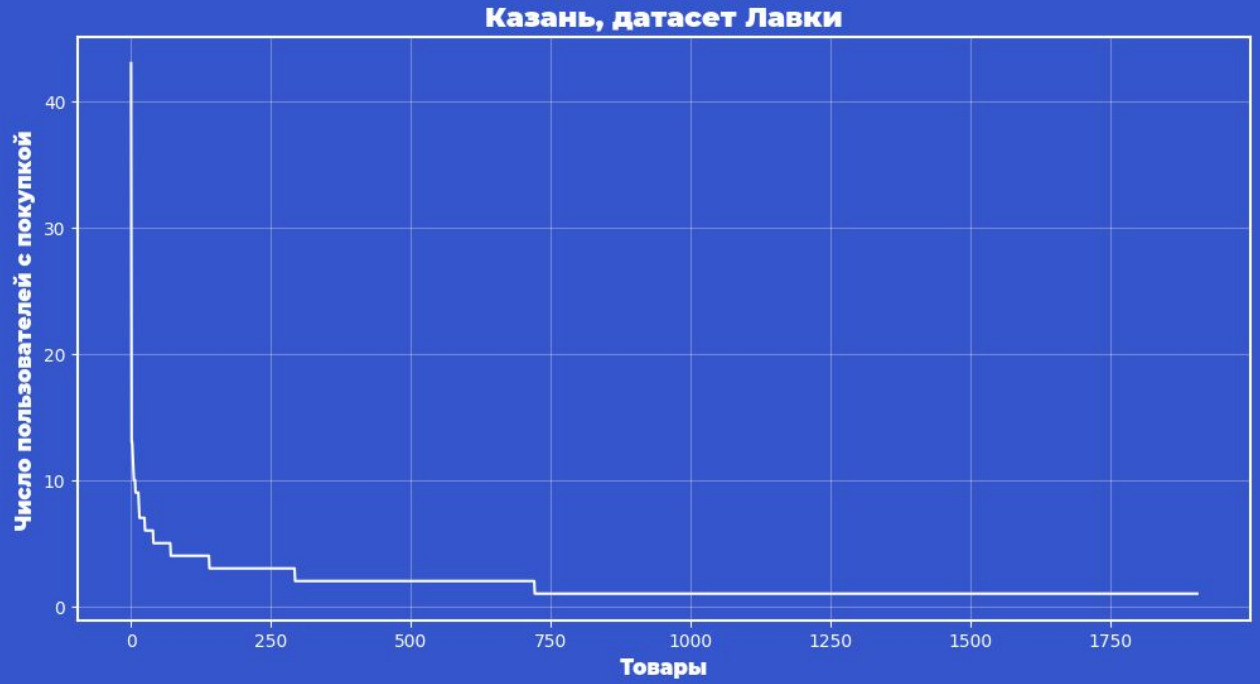
- Просто список разных айтемов, по чему-то отсортированный .
- В простом варианте строятся в оффлайне и грузятся на тачки настолько часто, насколько успеваете
- В сложном – можно построить отдельный рантайм-процессинг для поддержания айтемных статистик truthful в near-rt, а потом поверх него строить такие топы с похожими гарантиями
- В рантайме лежат в памяти и мгновенно используются для генерации кандидатов.
- При раскладывании на шарды здесь и далее к исходному множеству просто применяем фильтрацию по айтемам шарда

Статические топы

Например, по популярности
- клики
- покупки
- CTR, pCTR
- wCTR, wpCTR

и свежести
- новинки
- набирающие популярность

за разные периоды времени



Кворумы

Попытка при помощи топов сделать что-то персональное, основываясь на идее TF-IDF

Идея: пользователей будем считать “документами”, а их айтемы-позитивы – “термами”

Для того, чтобы конструкция переносилась на новые айтемы вне “обучения”, введем свертку по категориям айтемов, чтобы привычный TF-IDF стал чем-то в духе

$$\sum_{category} TF\,(item,\, category) UF\,(user,\, category) IDF\,(category)$$

Кворумы

$$\sum_{category} TF\left(item, category\right) UF\left(user, category\right) IDF\left(category\right)$$

Определим term-frequency просто как средний набор категорий среди пользователей его употреблявших, где $UF\left(user, category\right)$ – вес категории у пользователя

$$TF\left(item, category\right) = \frac{\sum_{user \in U(item)} UF\left(user, category\right)}{|U\left(item\right)|}$$

IDF определим более-менее обычным образом

$$IDF\left(category\right) = \min\left(-\log \overline{UF\left(user, category\right)}, \gamma\right)$$

Как тогда набрать айтемы по этой штуке, когда пришел пользователь:

- **Построим топы по TF по категориям в оффлайне, положим их на тачку**
- **На практике берут из каждого топа по N документов, считают для объединения эту функцию и берут топ**

KNN-индексы

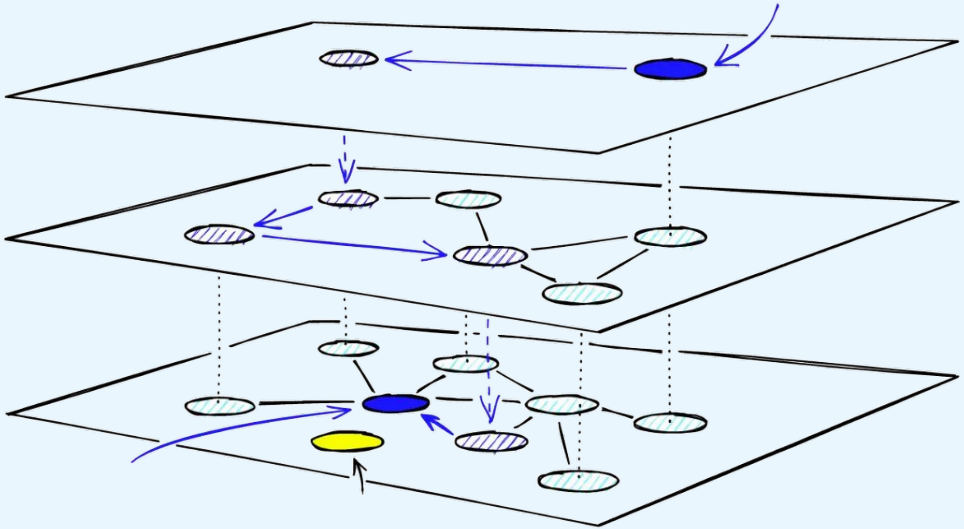
Идея, которая долгое время была (да и в основном остается) главной в кандидатогенерации .
При помощи моделей, который эмбедят пользователей и айтемы в одно пространство , можно погенерировать кандидатов так:

- Построить эмбединги всех айтемов
- Построить по ним структуру данных, которая за приемлемое время умеет по пользовательскому эмбеду находить ближайшие айтемы
- Положить ее на тачки
- В рантайме откуда-то брать пользовательский эмбед и по нему извлекать айтемы из этой структуры данных

Можно медленно искать точно (KNN-индексы) или быстро приближенно (ANN-индексы).
Мы посмотрим на ANN-индекс подробнее на примере HNSW.

картинки взяты с https://www.pinecone.io/learn/series/faiss/hnsw/ [1]

HNSW



Hierarchical Navigable Small World – структура данных, которая по точке в пространстве приближенно ищет N ближайших соседей из заранее заданного множества

Концептуально :

Построение

- **Все точки раскладываются в иерархическую систему слоев , каждый из которых – small-world-граф : между любыми двумя точками существует короткий путь (aka 6 рукопожатий)**

Поиск

- **Пока не дойдем до последнего слоя, жадно ищем ближайшую к целевой на слое, а потом проваливаемся по ней вниз**
- **На последнем слое перебираем чуть побольше для пущей точности**

HNSW
Препроцессинг

Берем наше множество точек и для каждой точки по очереди

- **Определяем уровни, на которые она попадет**
- **Максимальный уровень сэмплим из**

$$P(maxLevel = k) \sim \text{Geometric}(e^{-\lambda k}, levelCount) \text{ или}$$

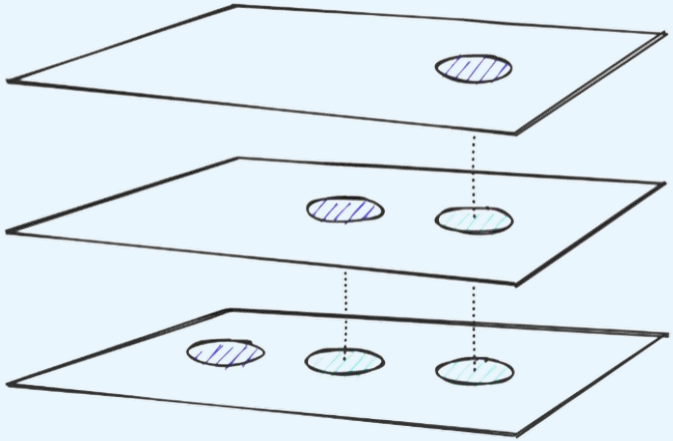
$$P(maxLevel = k) = e^{-\lambda k}(1 - e^{-\lambda k})^{levelCount}, \text{ где}$$

$$\lambda = \frac{1}{M}$$

- **Во все меньше него попадает автоматически. По сути просто какое-то красивое экспоненциальное замедление**
- **Здесь M – гиперпараметр, отвечающий за максимальную степень вершины в графе (увидим дальше)**

HNSW
Препроцессинг

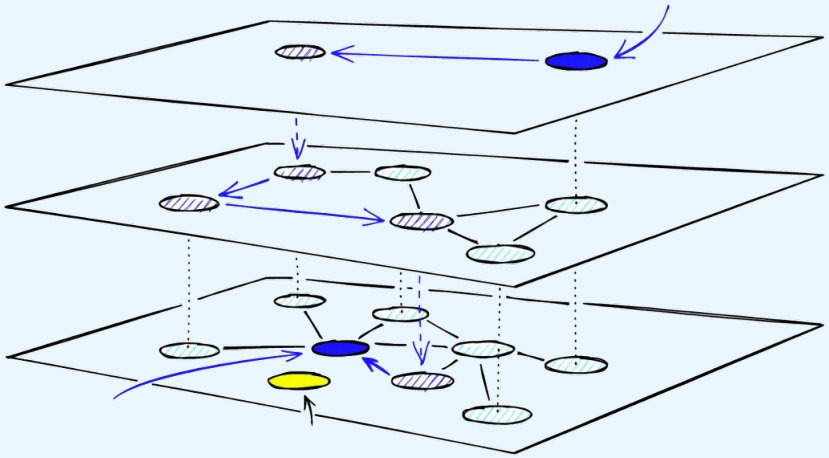
- Пока не дойдем до первого слоя, где есть точка , поддерживаем на слое жадного ближайшего соседа: на самом высоком слое берем честно посчитанного ближайшего, спускаемся на слой ниже по этой же вершине (она там есть по построению) и идем по ребрам, пока расстояние уменьшается (всегда выбираем жадно)



- На остальных слоях своего рода BFS-ом ищем M ближайших: у BFS'а есть максимальный размер очереди – efConstruction . При этом в очередь добавляются новые вершины только если они жадно уменьшает расстояние до новой точки. Соединяем эти M точек с нашей ребрами

HNSW
Поиск

- Стартуем с самого высокого слоя, на нем очень мало точек и можно просто руками найти ближайшую
- На каждом следующем слое идем от нее по вершинам-соседям, пока можем жадно уменьшить расстояние до искомой точки



- На самом последнем слое снова делаем странный BFS: ограничиваем размер очереди до efSearch . При этом в очередь добавляем новые вершины только если они жадно уменьшает расстояние до новой точки.
- По дороге поддерживаем heap с топ-N соседями, ими и отвечаем.

Посмотрим на примере Alternating Least Squares

Одна из **первых продуктивных идей** в рекомендациях: советовать вам то, что понравилось людям, которым понравилось то, что понравилось вам
Это еще называют **коллаборативной фильтрацией**

- Можно попробовать сделать как-то наивно, но популярнее делать иначе
- Построим **матрицу взаимодействий** и что-нибудь с ней сделаем
- Например, получим **эмбеды** айтемов и пользователей, как в **ALS [2]**
- Или прямо **скоры**, как в **SLIM [3]**

ALS

Он же – Alternating Least Squares

Идейно:

- **Хотим приблизить исходную матрицу в виде произведения двух**

$$R \approx \hat{R} = U \times V^T$$

- **В одной пользовательские эмбеды, в другой – айтемные**
- **Тогда для известных оценок можно выписать вот такой лоссик (сразу с bias'ами)**

$$L = \sum_{(u,i) \in R} (\hat{r}_{ui} - r_{ui} - b_u - b_i)^2 + \lambda_{emb} \left(\sum_{u \in U} \|p_u\|^2 + \sum_{iinI} \|q_i\|^2 \right) + \lambda_{bias} \left(\sum_{u \in U} b_u^2 + \sum_{i \in I} b_i^2 \right)$$

- **Можно поочередно "фризить" пользовательские и айтемные параметры и решать систему уравнений, которая получается при взятии производной**
- **Это метод наименьших квадратов, отсюда название**

ALS

То, что мы описали на предыдущем слайде – eALS, он же explicit ALS

- Поскольку лоссик считается по явным оценкам
- Что делать, если у нас неявный фидбек?

Вообще вариантов масса, но популярный подход – iALS, он же implicit ALS

- Все как с explicit'ом, но
- Лосс считается по всей матрице, предполагая неизвестные интеракции нулями

$$L = \sum_{i,j} C_{ij} \left(R_{ij} - U_i V_j^T \right) + \lambda \dots$$

- Добавляется " уверенность " в каждой оценке тем более большая, чем сама оценка

$$C_{ij} = 1 + \alpha R_{ij}$$

В кандидатогенерации

- Считаем пользовательские и айтемные эмбеды
- Пользовательские эмбеды кладем в какой-нибудь key-value storage
- В рантайме забираем из него эмбед пользователя
- На тачки раскладываем ANN-индекс
- Ищем ближайшие к пользовательскому эмбеду в ANN-индексе по такой близости, чтобы оптимизировала максимум (легко построить)

В ранжировании

- Используем скалярное произведение эмбедов пользователя и айтема как фичу в модели
- Нормы хорошо проксируют популярность на айтемах
- И горячесть на пользователях
- Либо прям сами векторы, если feeling adventurous или учим нейросетевое ранжирование

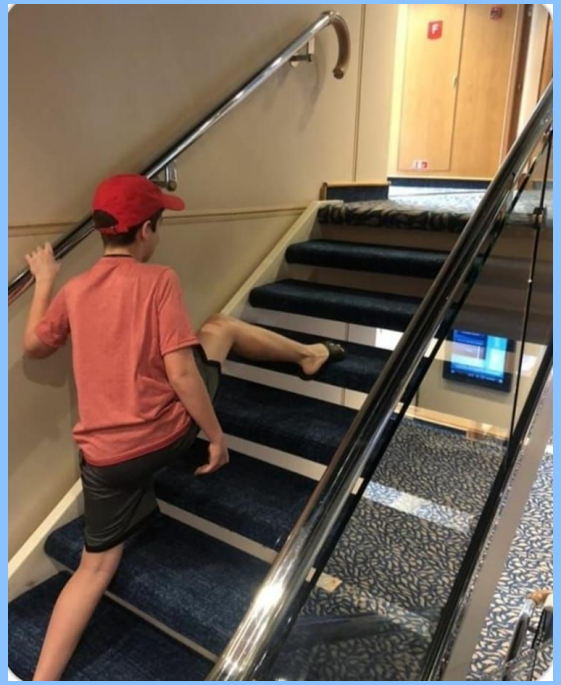
Об инженерных нюансах

Проблемки

- В обоих случаях есть неприятный минус :
  - Матрица строится для заранее определенного набора пользователей и айтемов
  - Что делать, если появляется новый айтем или пользователь ?
  - Базовый ответ: построим конвейер обучения
- Какие проблемы будут у такого конвейера?
  - Если учить с нуля, интуитивно кажется, что распределение фичи скалярного произведения может ехать

Пошагаем

- Если делать “шаги”, проблем особо не будет
- Допустим, хотим посчитать значения векторов для новых пользователей и айтемов
- Сначала посчитаем для новых айтемов
- Выберем последние вектора, которые у нас были для пользователей, и применим метод наименьших квадратов (один шаг)
- Теперь шагнем в обратную сторону к пользователям
- На практике полезно “ходить” только через пользователей и айтемы с большими нормами (e.g. “прогретые”)
- Это мало роняет качество, дает более стабильные решения и позволяет ходить очень быстро и с маленькими задержками рантайма при правильной выстроенной инфраструктуре



Перестанем шагать

- При хождении через горячих пользователей или айтемы можно немного отойти от оптимума
- Поэтому полезно иногда полностью перестраивать ALS с нуля
- Занимательный факт: фича скалярного произведения от этого на практике, как ни странно, ни едет
- И это можно просто ставить на конвейер
- Итого на практике мы имеем два конвейера
  - Один с переобучением с нуля
  - Другой с шагами в near realtime

Другие интересные моменты

Зачем еще нужны шаги

Если у вас есть контентный эмбед айтема , можно быстро получить матчащийся эмбед пользователя

Пусть у вас есть языковая модель, которая эмбедит айтем, тогда:

- Посчитаем соответствующие вектора айтемов
- Сделаем шаг ALS от них к пользователем
  - Зафиксируем вектора айтемов в задаче ALS равные векторам из языковой модели
  - Решим систему уравнений
- Это своего рода “ правильное усреднение ” с учетом коллаборативной специфики

Аналогично можно действовать, если у вас есть эмбеддинг или модель для пользователя

Когда и какой ALS жжет

- На практике iALS больше жжет в кандидатогенерации
  - Действительно, в лоссе сравниваем его со всеми айтемами вообще
- А eALS в фичах модели
  - Там сравниваем только с настоящими негативами
- Обычно хуже работает на холодных пользователях и айтемах
- Очень чувствительна к регуляризации , обязательно нужно много с ней экспериментировать

- **Все еще крайне сильный бейзлайн, несмотря на все продвижения в нейронных сетях**
- **В 2022 году на RecSys (главная конференция по рекомендательным системам) вышла [4] статья, демонстрирующая, что хорошо затюненный ALS на ключевых датасетах не уступает или мало уступает SotA-решениям**
- **В 2024 с HSTU[5]/TIGER[6]/Wukong[7] ситуация точно изменилась, когда в рекомендациях научили скейлить сетки, чтобы рос профит**
- **На практике ALS – хорошая стартовая (а для многих – и неплохая финишная) точка для развития рекомендаций в вашем продукте**

Бизнесовые кандгены

Не упомянули о них сразу, но на практике очень важны и работают прекрасно. Общий вайб:

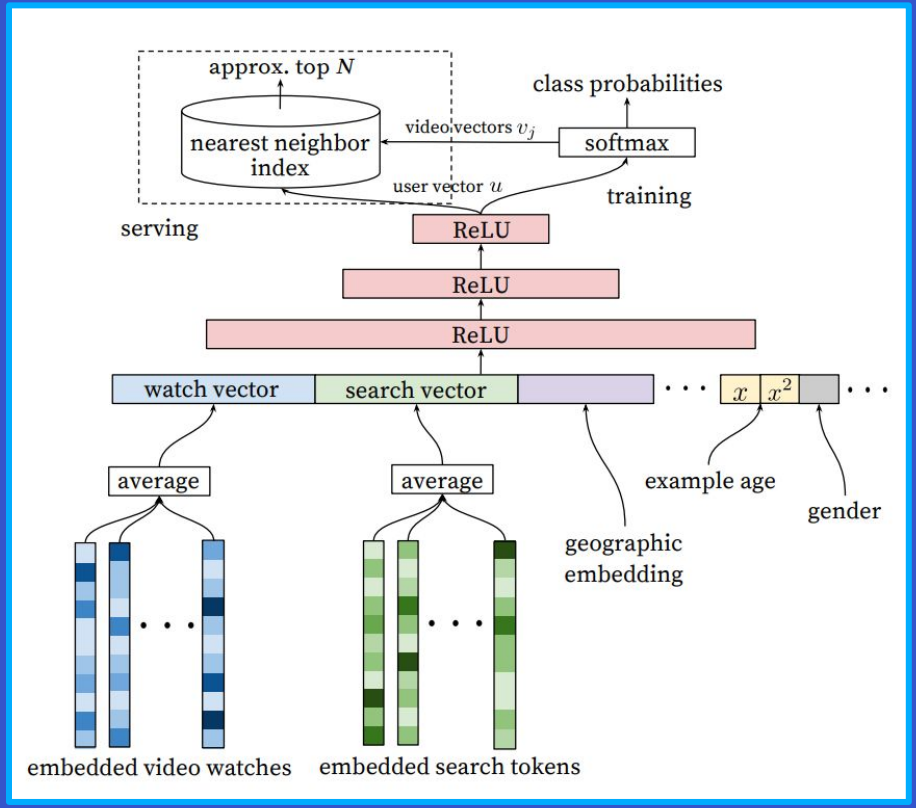
- история покупок
- история прослушиваний
- текущая корзина
- плейлисты пользователя
- избранное
- разные топы по близости к ним
  - из KNN-индекса по какому-то эмбеду
  - NPMI
  - отдельная модель на близость
  - you name it

KNN-индексы

- долгое время были +- единственным вариантом использовать нейронные модели для рантайм-кандидатогенерации
- отсюда много архитектур с 2 башнями : юзерской и айтемной
- про нейронки для кандидатогенерации будет отдельная лекция
- но глянем на классический пример из 2016ого [8]

YoutubeDNN

- Статья гугла
- Уже 2 башни, но айтемная башня плоская и не упоминается
- На практике все делали ее сложнее
- Attention is as all you need еще не вышла, просто усредняем айтемы из истории
- Задача мультиклассификации
- В софтмаксе случайные негативы с коррекцией популярности
- Как конвейеризовывали:
  - не использовали скор в ранжировании и переобучали с нуля
  - либо никак
  - остальное ложь, не верьте



KNN-индексы

- сейчас отдельные богатые компании поняли, что видеокарты умеют круто считать близости между огромным количеством эмбеддингов, при этом близости более сложные, чем косинус, и это интересное направление рисерча
- на примере MoL [9] и других статей посмотрим дальше в курсе
- HNSW и другие ANN-индексы строятся довольно медленно на большом множестве айтемов
- из-за этого они не решают проблем свежести, новизны и всего такого
- как быстро добавлять такие айтемы к нам в систему, если вы не настолько богатая компания?

i2i-списки

Обучим очень крутую i2i-модель:

- возьмем все контентное, что сможем
- поднимем крауд на похожесть
- обучим под его скоры модель

Теперь к каждому айтему можно:

- построить топчик по похожести (которая чисто контентная)
- переранжировать неперсональной контентной моделью целевого действия

В рантайме будем брать позитивы из истории, выбирать из них разнообразные (про алгоритмы разнообразия расскажут в следующей лекции) и по ним лукапить топы списков для этапа ранжирования.

Если айтемов слишком много, можно оставить только некоторые (самые репрезентативные).

Это в проде у Дзена [10]. Это же (но на существенных нейросетевых стероидах с меньшим Inductive bias) в проде у bytedance (tiktok) – RQ VAE [11] будет дальше в курсе.

Спасибо!

Даня Ткаченко,
Служба ML-сервисов Лавки,
Белград 2025

[1] Hierarchical Navigable Small Worlds (HNSW)
[2] Collaborative Filtering for Implicit Feedback Datasets
[3] SLIM: Sparse Linear Methods for Top-N Recommender Systems
[4] Revisiting the Performance of iALS on Item Recommendation Benchmarks
[5] Actions Speak Louder than Words: Trillion-Parameter Sequential Transducers for Generative Recommendations
[6] Recommender Systems with Generative Retrieval
[7] Wukong: Towards a Scaling Law for Large-Scale Recommendation
[8] Deep Neural Networks for YouTube Recommendations
[9] Retrieval with Learned Similarities
[10] Переосмысление item2item-рекомендаций в дзене
[11] Vector Quantization for Recommender Systems: A Review and Outlook