ОКН ВЫСШАЯ ШКОЛА ЭКОНОМИКИ

Классические бейзлайны и Yambda

Нейросетевые рекомендательные системы
2025/26, семинар 2

Владимир Байкалов

Ведущий разработчик,
Отдел Исследований
VK

Яндекс

План занятия

Про валидацию:

1. Немного про возможные сплиты

Классические бейзлайны:

1. Проверки базового смысла
2. DecayPop + EMA
3. ALS
4. ItemKNN
5. SLIM
6. EASE
7. SANSA

Практическая часть с кодом

Про валидацию

Часть 1

Про валидацию моделей

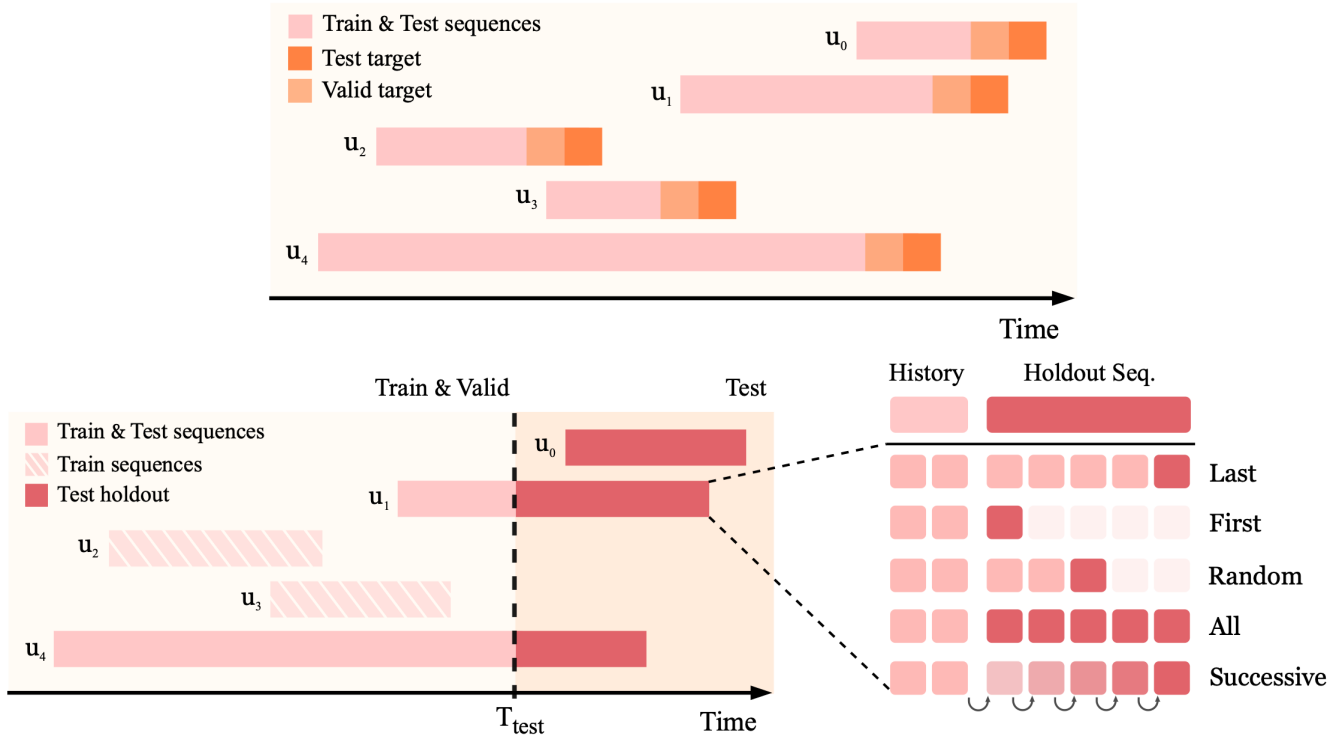
В статьях используется **leave-one-out**

Он порождает **утечки в данных**

Самый близкий к проду сценарий
Global Temporal Split

В Yambda у нас был **All** сценарий

Есть ряд статей[1,2] которые также поднимают проблему утечки



Time to Split: Exploring Data Splitting Strategies for Offline Evaluation of Sequential Recommenders
Gusak et al.

[1] A Critical Study on Data Leakage in Recommender System Offline Evaluation
[2] Exploring Data Splitting Strategies for the Evaluation of Recommendation Models

Классические бейзлайны

Часть 2

Преамбула

Зачем нам изучать что-то кроме нейронных сетей?

- Полезно знать что было до нас
  - Понимание развития области будет более связным
- Рекомендательные системы это не только нейронки
  - Не всегда стоит начинать именно них
  - Они не являются “серебряной пулей”
  - Много проблем можно решить классическими методами

Yambda-5B	Random
	MostPop
	DecayPop
	ItemKNN
	iALS
	BPR
	SANSA
	SASRec

Yambda-5B — A Large-Scale Multi-modal Dataset for Ranking And Retrieval
Ploshkin et al.

Преамбула

На каждой стадии применятся свои подходы

- Сегодня в основном будем говорить только про стадию отбора (**Retrieval**)
- Хотим чтобы наши алгоритмы выдавали топ-**K** айтемов для пользователя
- Некоторые подходы можно применять и для ранжирования (**Ranking**)
  - Об этом будем проговаривать явно

Проверки здравого смысла (sanity checks)

Random

- Выдаем пользователю **К случайных айтемов**

Popular

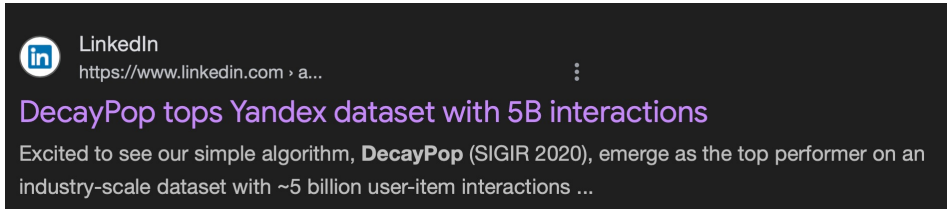
- Считаем сколько раз каждый айтем встретился в обучающей выборке
- Выдаем **всем** пользователям пользователю **одинаковый** топ-К по популярности

Еще немного про **Popular**:

- Уже довольно сильный алгоритм
- Какие у него есть проблемы?

DecayPop

Улучшаем подход на 70% (по словам авторов статьи)



Ссылка на пост

У нас в сервисах есть **Distribution Shift**

- Распределения **постоянно меняются**
  - появляются новые тренды, обновляется каталог
  - меняются интересы пользователей
- Нужны алгоритмы, которые **постоянно дообучаются** на новых данных

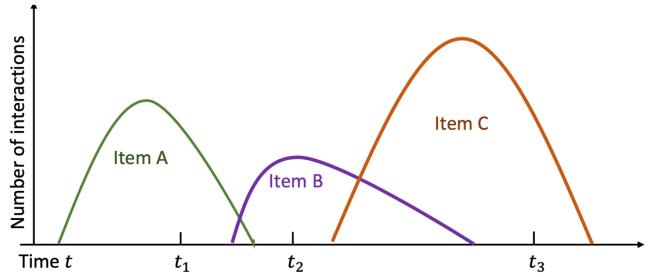


Figure 1: Popularity of items A, B, and C over time.

A Re-visit of the Popularity Baseline in Recommender Systems

DecayPop

MostPop: $$P_i = \sum_{(u,i)} \mathbb{I}\left[r_{u,i} \neq 0\right]$$

RecentPop: $$P_i = \sum_{u \in \mathcal{U}} \sum_{(i,t) \in S_u} \mathbb{I}\left[t \geq t^{\dagger}\right]$$

DecayPop: $$P_i = \sum_{u \in \mathcal{U}} \sum_{(i,t) \in S_u} e^{-|t_{max} - t|}$$

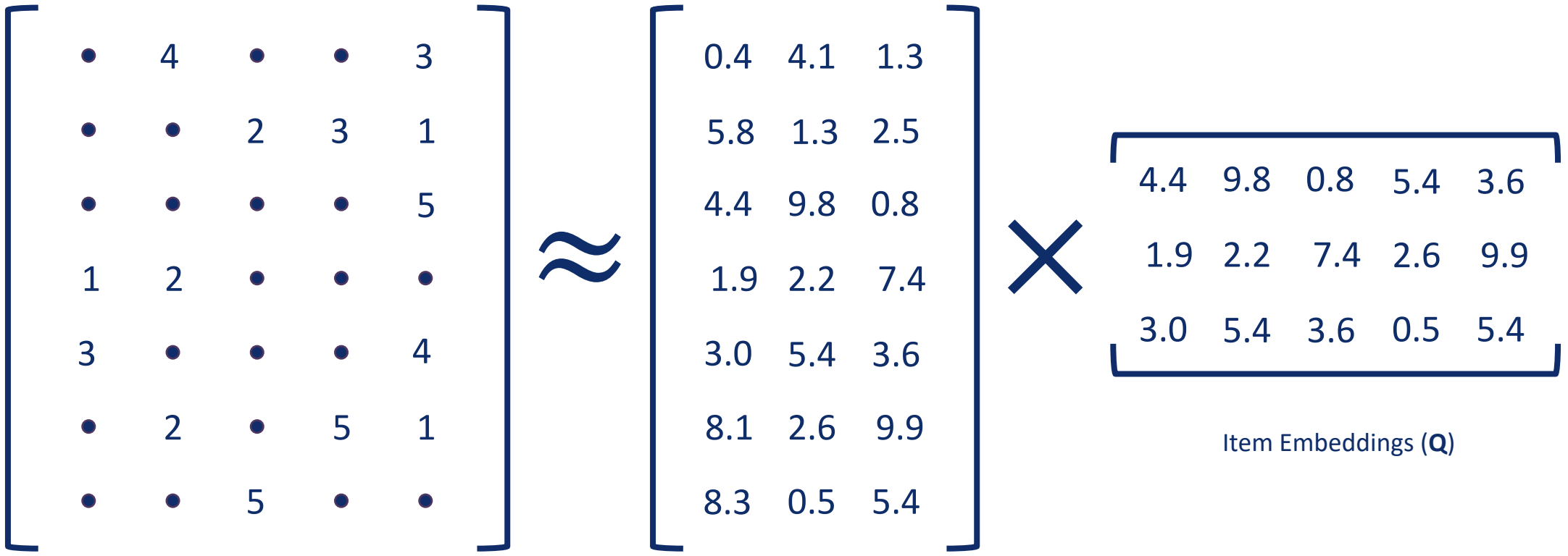
Table 1: Results of popularity methods; best results in bold

Popularity	HR@5	HR@10	NDCG@5	NDCG@10
MostPop	0.0304	0.0462	0.0198	0.0248
RecentPop	0.0530	**0.0845**	0.0338	0.0440
DecayPop	**0.0532**	0.0843	**0.0341**	**0.0441**

[1] A Re-visit of the Popularity Baseline in Recommender Systems

Матричная факторизация

Представим user-item матрицу как произведение двух низкоранговых матриц:



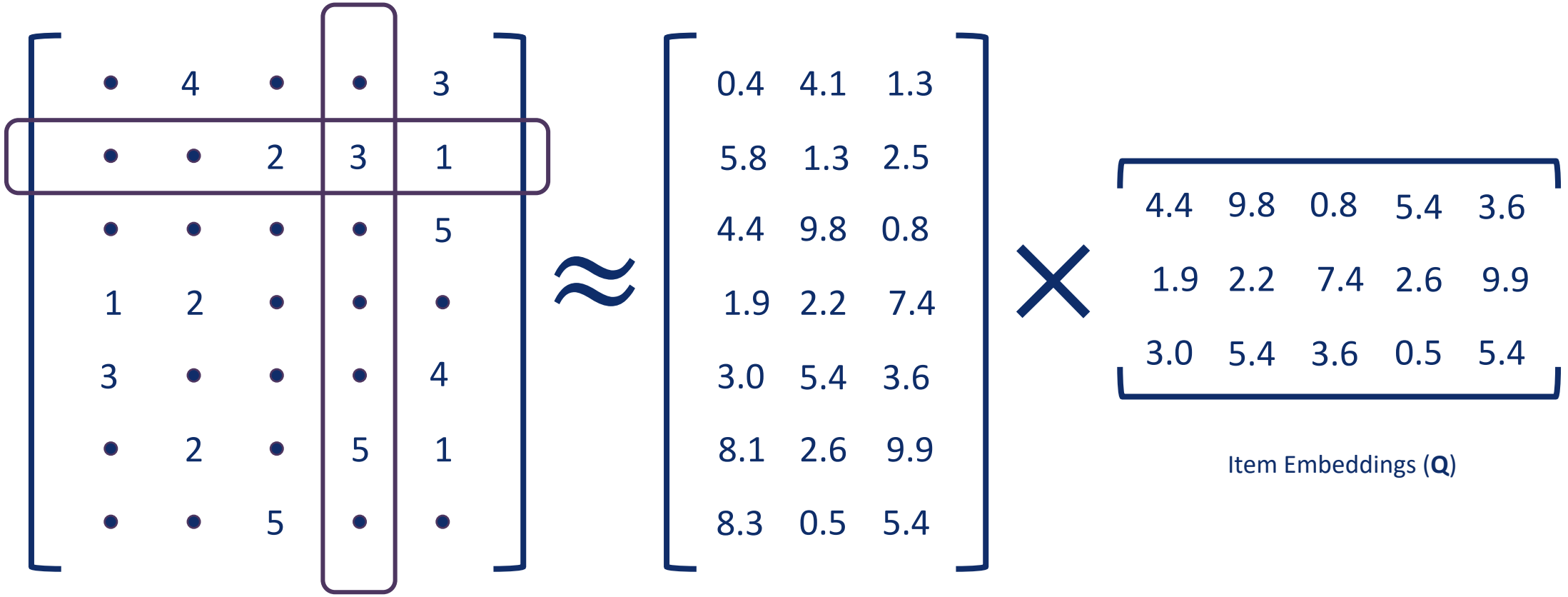
User-item interaction matrix (**R**)

User Embeddings (**P**)

Item Embeddings (**Q**)

Матричная факторизация

Представим user-item матрицу как произведение двух низкоранговых матриц:



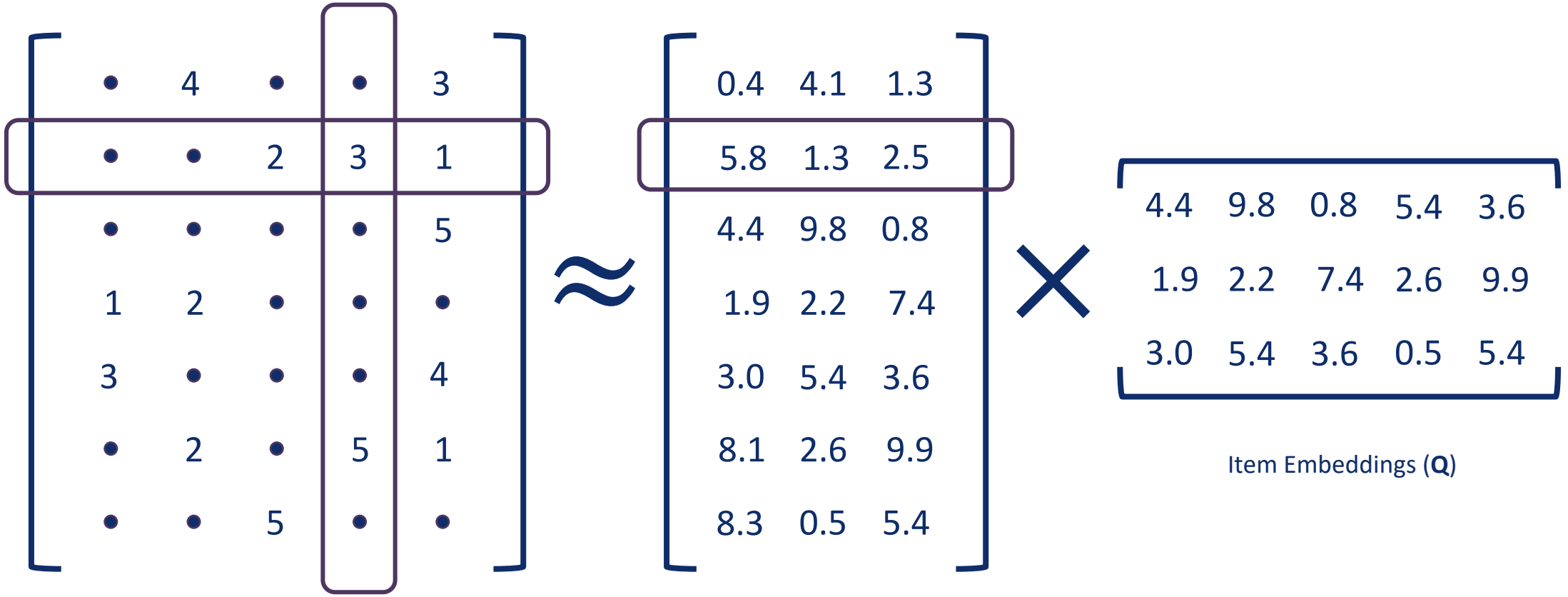
User-item interaction matrix (**R**)

User Embeddings (**P**)

Item Embeddings (**Q**)

Матричная факторизация

Представим user-item матрицу как произведение двух низкоранговых матриц:



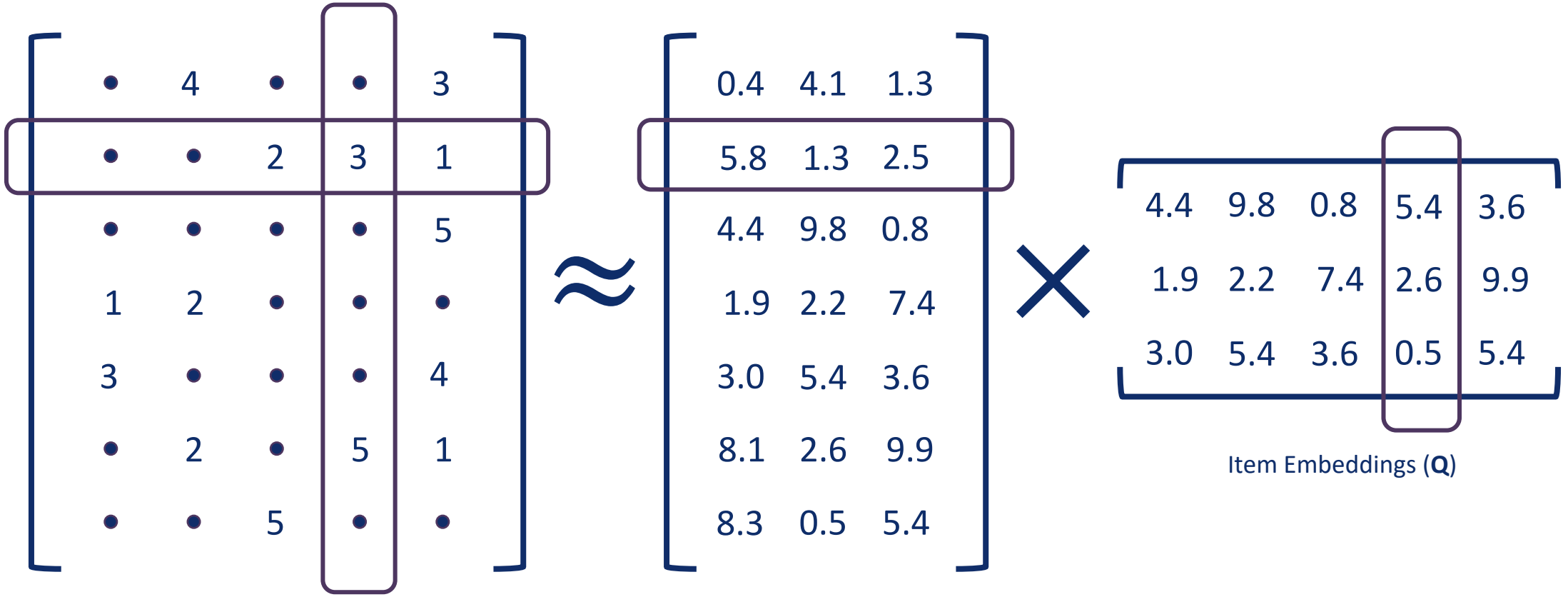
User-item interaction matrix (**R**)

User Embeddings (**P**)

Item Embeddings (**Q**)

Матричная факторизация

Представим user-item матрицу как произведение двух низкоранговых матриц:



User-item interaction matrix (**R**)

User Embeddings (**P**)

Item Embeddings (**Q**)

Матричная факторизация

Получить векторные представления пользователей и объектов на основе данных об их взаимодействиях.

Функция оценки релевантности для пар (пользователь, объект)

$$f_{(u,i)} : \text{User}, \text{Item} \to \text{Relevance}$$

Вектора получаем за счет фиксирования какой-либо функции оценки и решения задачи оптимизации

$$\mathcal{L}(R, \hat{R}) \to \min$$

Вопрос: почему это плохо делать?

ALS

Одна из первых продуктовых идей

$$L = \sum_{(u,i)} c_{u,i}\left(r_{u,i} - p_u q_i^\top\right)^2 + \lambda\left(\sum_u \|p_u\|^2 + \sum_i \|q_i\|^2\right)$$

$$L = \left\|C * \left(R - PQ^\top\right)\right\|_F^2 + \lambda\left(\|P\|_F^2 + \|Q\|_F^2\right)$$

Плюсы ALS:

- Хорошо параллелится,
  (у вас про это было/будет на **LSML** курсе)
- Очень мощный и простой алгоритм
- Применяется повсеместно
- Нормы коррелируют с популярностью пользователей и айтемов

$$\frac{\partial L}{\partial p_u} = 0 \Rightarrow p_u = \left(Q^\top Q + Q^\top (C^u - I)Q + \lambda I\right)^{-1} Q^\top C^u r_u,$$

$$\frac{\partial L}{\partial q_i} = 0 \Rightarrow q_i = \left(P^\top P + P^\top (C^i - I)P + \lambda I\right)^{-1} P^\top C^i r_i.$$

ALS. Implicit vs Explicit

По каким парам обучаться?

- По существующим парам пользователь айтем для которых известна оценка (**explicit**)
- По всем возможным парам (**implicit**)

ALS. Применение в проде

Считаем пользовательские/айтемные эмбеды

Стадия отбора:

Айтемные кладем в векторный индекс

По запросу пользователя идем с его вектором в индекс

Стадия ранжирования:

В ранжировании просто как фичу скаляр

Где какой ALS используется:

Отбор – **implicit**

Ранжирование – **explicit**

ALS. Дообучение

Проблема:

Матрица строится для заранее определенного набора пользователей и айтемов

Что делать в случае новых объектов?

Делаем шаг ALS’а для новых пользователей, а затем для новых айтемов

$$p_u = \left(Q^\top Q + Q^\top \left(C^u - I\right)\hat{Q} + \lambda I\right)^{-1} Q^\top C^u r_u.$$

Обычно все равно **требуется полное переобучение** ALS’а

ItemKNN

Интуиция:

Пользователь – набор айтемов

Рекомендации – выдача айтема, похожего на существующие взаимодействия

$$W_{i,j} = \mathrm{sim}(R_{:,i}, R_{:,j}) \qquad \hat{r}_{u,i} = \sum_{j \in S_u} R_{u,j}\, W_{j,i}$$

Применение:

1. Посчитать **W** заранее
2. Для каждого из айтемов топ-**K** соседей для каждого айтема
3. При запросе пользователя брать похожести только с этими айтемами, сложность **O(LK)**, где L число событий пользователя

Проблемы ItemKNN

Мы не обучались на целевую задачу приближения **R**

Описывать айтем похожими по какой-то близости это эвристика

Идея:

Сделать матрицу **W** обучаемой, но использовать ту же линейную форму

$$\hat{R} = RW \qquad \hat{r}_{u,j} = \sum_{i \neq j} r_{u,i} W_{i,j}$$

SLIM. Sparce LInear Method

Идея:

Давайте аппроксимировать матрицу взаимодействий (**R**), не напрямую, линейную комбинацию ”похожих айтемов”.

$$\hat{R} = RW \qquad \hat{r}_{u,j} = \sum_{i \neq j} r_{u,i} W_{i,j}$$

Ограничения:

На диагонали **W** должны **стоять нули**

Хотим, чтобы $\mathbf{W_{ij}}$ были **неотрицательны**

Решение: $\min_{W} \|R - RW\|_F^2 + \lambda_1 \|W\|_1 + \lambda_2 \|W\|_F^2 \quad \text{s.t.} \quad \text{diag}(W) = 0, \; W \geq 0.$

[1] SLIM: Sparse Linear Methods for Top-N Recommender Systems

SLIM. Sparce LInear Method

Основная проблема SLIM:

Медленное обучение, из-за **L_1 регуляризации**

Нет аналитического решения

EASE – Решение этой проблемы

[1] SLIM: Sparse Linear Methods for Top-N Recommender Systems

EASE. EmbArrassingly Shallow autoEncoder

Чем хорош SLIM:

Получаем **разреженный W**

Есть интерпретация “похожести”

$$\hat{R} = RW, \qquad \hat{r}_{u,j} = R_{u,:}\, W_{:,j}$$

$$\min_{W} \ \|R - RW\|_F^2 + \lambda \|W\|_F^2 \quad \text{s.t.} \quad \operatorname{diag}(W) = 0$$

Основная проблема SLIM:

Медленное обучение, из-за **L_1 регуляризации**

$$G = R^\top R, \qquad P = (G + \lambda I)^{-1}$$

Важное наблюдение:

Около **60% весов W** получаются **отрицательными**

Модель учит не только похожесть, но и непохожесть

$$W_{ij} = \begin{cases} 0, & i = j, \\ -\dfrac{P_{ij}}{P_{jj}}, & i \neq j. \end{cases}$$

[1] Embarrassingly Shallow Autoencoders for Sparse Data

SANSA. ScalAble Non-Symmetric Autoencoder

Проблема EASE:
Получение матрицы **P**
Матрица получается **плотной**, даже если **R разреженный**
EASE ограничен по сложности **сотнями тысяч айтемов**

$$G = R^\top R, \qquad P = (G + \lambda I)^{-1}$$

$$W_{ij} = \begin{cases} 0, & i = j, \\ -\dfrac{P_{ij}}{P_{jj}}, & i \neq j. \end{cases}$$

Идея SANSA:
Применить **Cholesky decomposition** для матрицы **P**

W представляется как произведение двух матриц:
энкодера (**E**) и декодера (**Z**)

$$P = A^{-1} \qquad A^{-1} \approx K^\top \hat{D}^{-1} K \qquad W \approx E^\top Z \qquad E = KP \qquad Z = \hat{D}^{-1} K$$

[1] Scalable Approximate NonSymmetric Autoencoder for Collaborative Filtering

Перейдем к коду

Часть 3