

LSML #5

Рекомендательные системы

Рекомендательные системы

Вам также могут понравиться



Tefal GV7485

от 13 790 руб.

0 отзывов 6 предложений

Утюг

- с парогенератором
- мощность 2000 Вт
- мощный паровой удар
- мощность подачи пара до 120 г/мин



Цены



Philips GC 1026

от 1 448 руб.

0 отзывов 94 предложения

Утюг

- мощность 2000 Вт
- мощность подачи пара до 25 г/мин
- вес 0.9 кг
- паровой удар



Цены



Tefal FV2352E0

от 1 800 руб.

1 отзыв 9 предложений

Утюг

- мощность 2000 Вт
- мощность подачи пара до 30 г/мин
- вес 1.4 кг
- паровой удар



Цены



Sinbo SSI-2872

от 900 руб.

1 отзыв 94 предложения

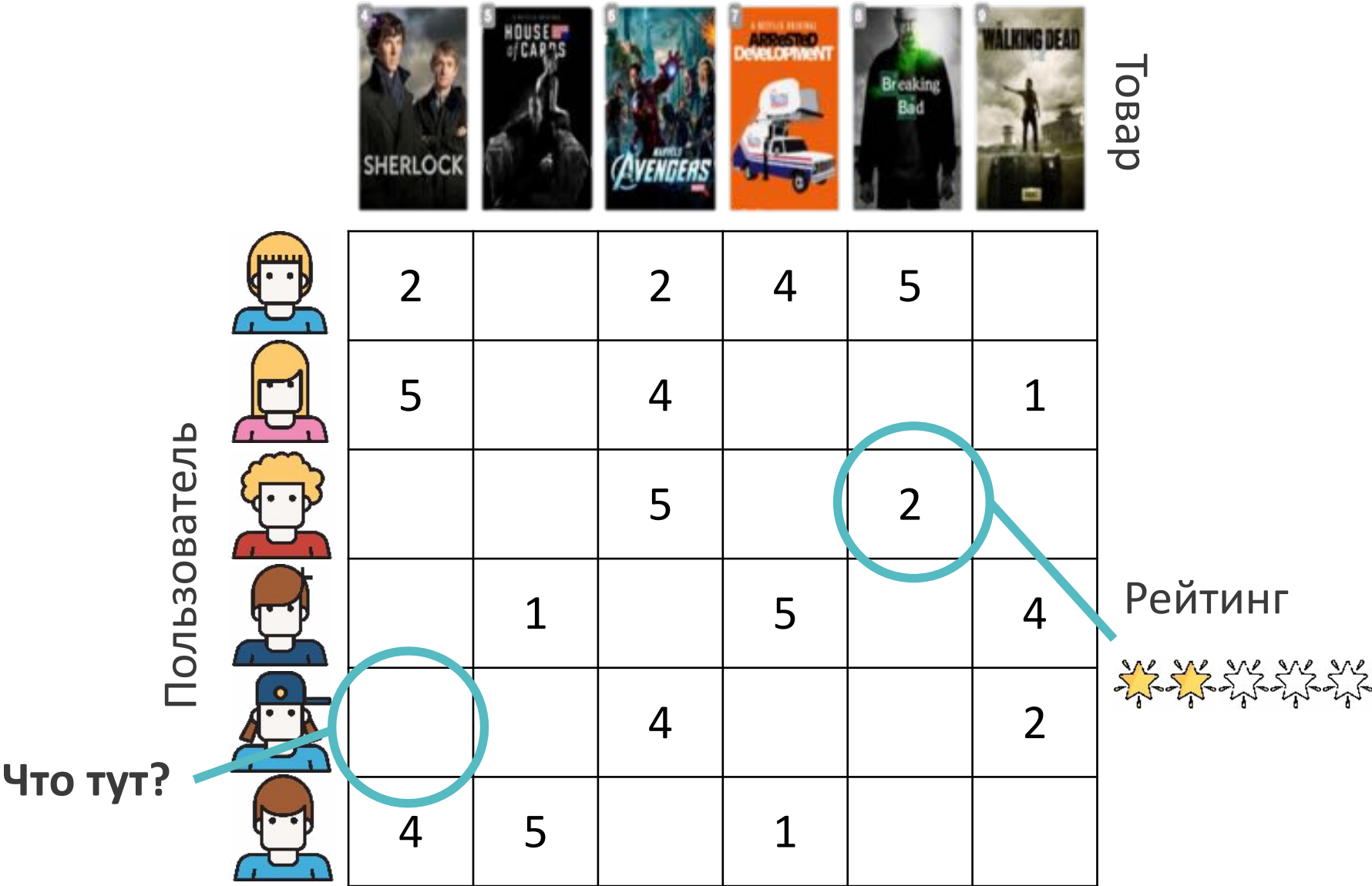
Утюг

- мощность 2000 Вт
- керамическая подошва
- паровой удар
- вертикальное отпаривание

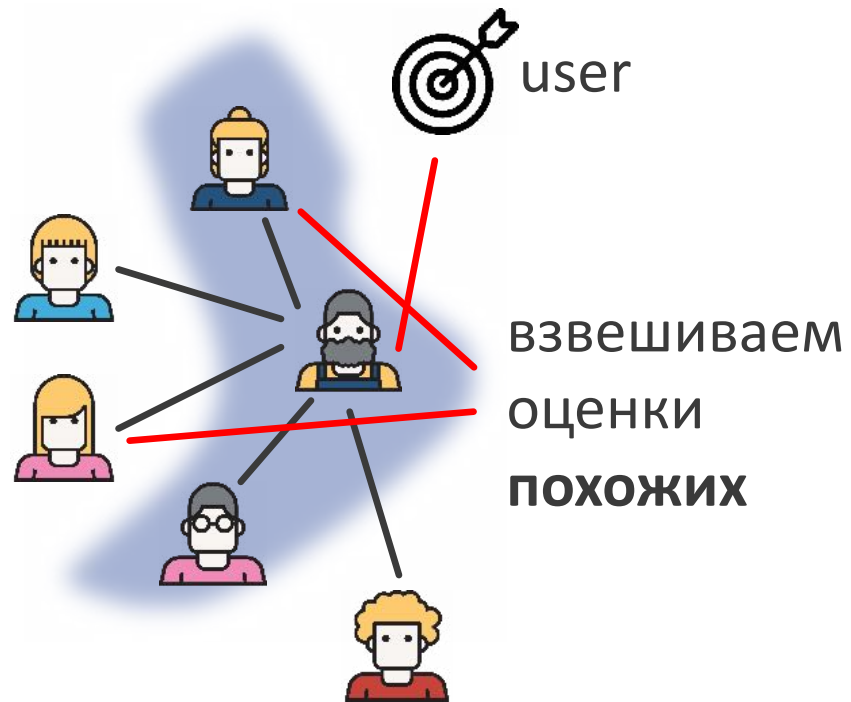


Цены

Рекомендательные системы











User-based и Item-based Collaborative Filtering



Похожесть пользователей

						
	2		2	4	5	
	5		4			1
			5		2	
		1		5		4
			4			2
	4	5		1		



						
	2		2	4	5	
	5		4			1
			5		2	
		1		5		4
			4			2
	4	5		1		



Корреляция
общих
оценок = -1

User-based CF

- n пользователей, m товаров
- r_{ui} — оценка пользователя u товару i
- r_u — средняя оценка пользователя u
- Похожесть пользователей u и v :

По общим оценкам!

$$corr(u, v) = \frac{\sum_{i \in I} (r_{ui} - r_u)(r_{vi} - r_v)}{\sqrt{\sum_{i \in I} (r_{ui} - r_u)^2 \sum_{i \in I} (r_{vi} - r_v)^2}}$$

- Предсказание:

$$\widehat{r_{ui}} = r_u + \frac{\sum_{v=1}^n corr(u, v)(r_{vi} - r_v)}{\sum_{v=1}^n |corr(u, v)|}$$

Прикинем на примере

- 10000 рейтингов
- 1000 пользователей
- 100 товаров
- Рейтинги распределены равномерно в таблице
- **Сколько общих оценок в среднем у двух случайных юзеров?**
- **А у двух случайных товаров?**

Прикинем на примере

- 10000 рейтингов
- 1000 пользователей
- 100 товаров
- Рейтинги распределены равномерно в таблице
- **Сколько общих оценок в среднем у двух случайных юзеров? (1)**
- **А у двух случайных товаров? (10)**

Item-based CF

- Похожие формулы
- Более уверенные оценки похожести товаров
- Медленнее устаревают похожести товаров
- В production:
 - Посчитаем для каждого товара в офлайне 1000 самых похожих на него, сложим в Key-Value хранилище (например, Redis)
 - В онлайнe возьмем оценённые пользователем товары (последние 100) и для них сходим в Key-Value за похожими
 - Посчитаем итоговые рекомендации по формуле $\hat{r}_{ui} = \frac{\sum_j s(i,j)r_{uj}}{\sum_j |s(i,j)|}$

Параллелим Item-based CF













- n пользователей, m товаров, $n \sim m \sim 10^6$
- Нужно посчитать заранее Item-Item похожести:

$$s(i, j) = \frac{\sum_{u \in U} (r_{ui} - r_u)(r_{uj} - r_u)}{\sqrt{\sum_{u \in U} (r_{ui} - r_u)^2} \sqrt{\sum_{u \in U} (r_{uj} - r_u)^2}} \quad \text{adjusted cosine similarity}$$

- Наивный подход – $O(m^2n)$
- Матрица оценок сильно разрежена, можно лучше?
- Вклад в похожесть двух товаров ненулевой только от пользователей, которые оценили оба этих товара

Инвертированный индекс

- Будем обновлять расчеты раз в день при помощи инвертированного индекса:

						
	2		2	4	5	
	5		4			1
			5		2	
		1		5		4
			4			2
	4	5		1		

$$s(i, j) = \frac{\sum_{u \in U} (r_{ui} - r_u)(r_{uj} - r_u)}{\dots}$$

Map шаг:

$(1, 3) \rightarrow (5 - 3.3) * (4 - 3.3)$

$(1, 6) \rightarrow (5 - 3.3) * (1 - 3.3)$

$(3, 6) \rightarrow (4 - 3.3) * (1 - 3.3)$

Reduce шаг:

Суммируем по ключу

Как посчитать все числители на Spark

```
def emit_pairs(x):
    user, items = x
    items = sorted(items)
    if len(items) < 300:
        for i in range(len(items) - 1):
            for j in range(i + 1, len(items)):
                yield (
                    (items[i][0], items[j][0]),
                    items[i][1] * items[j][1]
                )

dot_product = (
    ratings
    .map(lambda x: (x.user, (x.product, x.rating)))
    .groupByKey()
    .flatMap(emit_pairs)
    .reduceByKey(lambda x, y: x + y)
)
```

$$s(i, j) = \frac{\sum_{u \in U} (r_{ui})(r_{uj})}{\dots}$$

Решение на Spark SQL

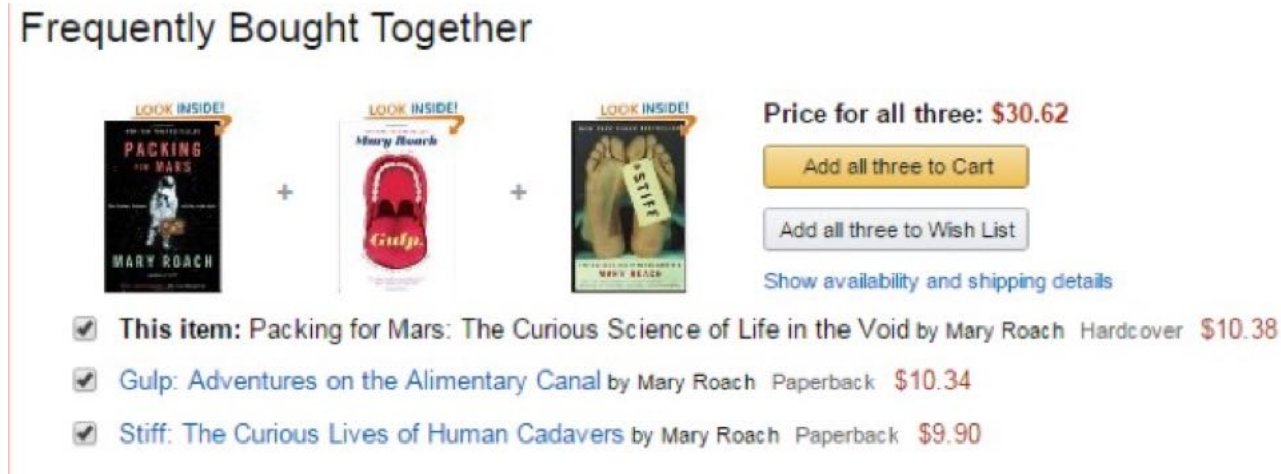
$$s(i, j) = \frac{\sum_{u \in U} (r_{ui})(r_{uj})}{\dots}$$

```
SELECT
  ic1.itemId,
  ic2.itemId AS jointItemId,
  SUM(ic1.val * ic2.val) AS cosine
FROM ic AS ic1
  JOIN ic AS ic2 ON ic1.clientId = ic2.clientId
WHERE ic1.itemId < ic2.itemId
GROUP BY ic1.itemId, ic2.itemId;
```

Похожести для неявных рейтингов

- Например, рассмотрим покупку (неявный рейтинг)

Frequently Bought Together



The screenshot shows three book covers: 'Packing for Mars', 'Gulp', and 'Stiff'. They are separated by plus signs. To the right, the total price is \$30.62. Below the books, there are two buttons: 'Add all three to Cart' (yellow) and 'Add all three to Wish List' (grey). A link 'Show availability and shipping details' is also present. At the bottom, a list of items with checkboxes and prices is shown.

Price for all three: **\$30.62**

[Add all three to Cart](#)

[Add all three to Wish List](#)

[Show availability and shipping details](#)

- ☒ **This item:** Packing for Mars: The Curious Science of Life in the Void by Mary Roach Hardcover **\$10.38**
- ☒ Gulp: Adventures on the Alimentary Canal by Mary Roach Paperback **\$10.34**
- ☒ Stiff: The Curious Lives of Human Cadavers by Mary Roach Paperback **\$9.90**

Что такое «часто покупают вместе»?

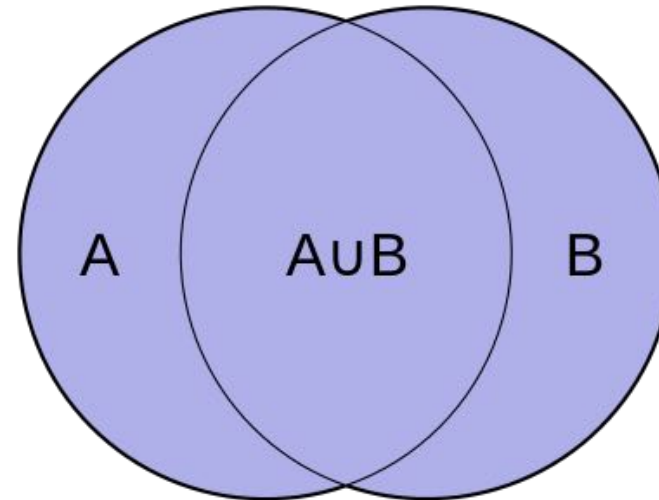
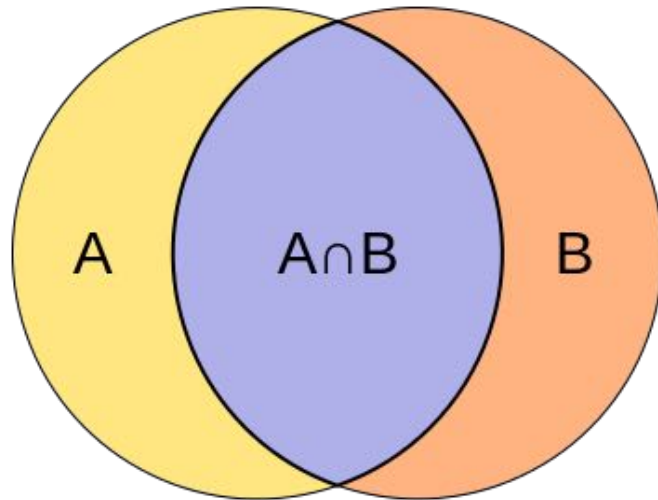
Количество совместных покупок?


- Бестселлеры типа «Гарри Поттер» много кто купил
- Получается, что «Гарри Поттер» часто покупают со всеми остальными книгами
- Но не рекомендовать же всем «Гарри Поттер» к любой книге?

Мера Жаккара (похожесть множеств юзеров)

- Учитывает популярности сравниваемых товаров:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$



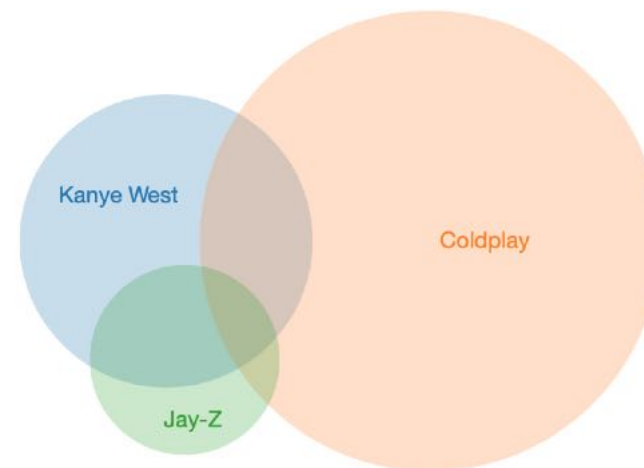
 – пользователи, купившие товар A

Пример

Самые похожие на Kanye West по пересечению:

Artist	Overlap
Coldplay	8061
Jay-Z	6851
The Beatles	6139
Radiohead	6135
Eminem	5454

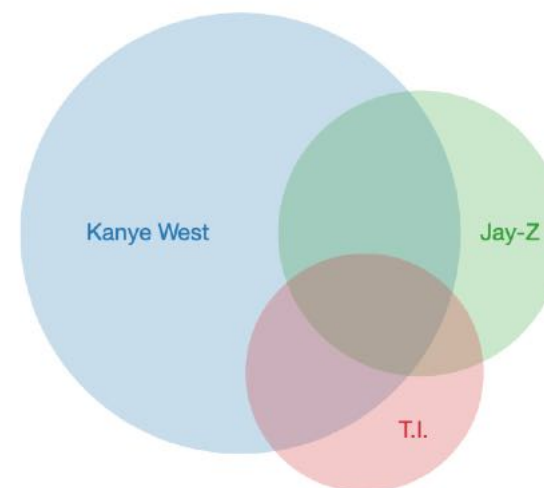
[more ...](#)



Самые похожие на Kanye West по мере Жаккара:

Artist	Jaccard
Jay-Z	0.217
T.I.	0.163
Lupe Fiasco	0.156
Nas	0.156
Common	0.139

[more ...](#)



Параллелится точно так же

	1	2	3	4	5	6
1	1		1	1	1	
2	1		1			1
3			1		1	
4		1		1		1
5			1			1
6	1	1		1		

$$J(A, B) = \frac{|A \cap B|}{\dots}$$

Map шаг:

$(1, 3) \rightarrow 1$

$(1, 6) \rightarrow 1$

$(3, 6) \rightarrow 1$

Reduce шаг:

Суммируем по ключу

Content-based рекомендации

- Используем свойства товаров и пользователей
 - Возраст, пол, ...
 - Жанр, режиссер, ...
- Можем рекомендовать товары, не имея статистики

Content-based: похожесть текстов

TFIDF Normed Vectors	a	Accelerating	and	applications	art	behavior	Building	Consumer	CRM	customer	data	for	Handbook	Introduction	Knowledge	Management	Marketing	Mastering	mining	of	relationship	Research	science	technology
Building data mining applications for CRM				0.502			0.502		0.344		0.251	0.502							0.251					
Accelerating customer relationships: using CRM and relationship technologies		0.432	0.296						0.296	0.216											0.468			0.432
Mastering Data Mining: the art and science of Customer Relationship Management			0.256		0.374					0.187	0.187					0.256		0.374	0.187	0.374	0.256		0.374	
Data Mining your website											0.316								0.316					
Introduction to Marketing														0.636			0.436							

Content-based: похожесть текстов

- Похожесть как косинус между векторами TF-IDF
- Используем инвертированный индекс

Факторизация матрицы оценок

- Матрица рейтингов **разреженная** (много пропусков)
- Оценки товаров **коррелируют** (пользователям часто нравятся одни и те же пары товаров)
- Можно сжать матрицу оценок!

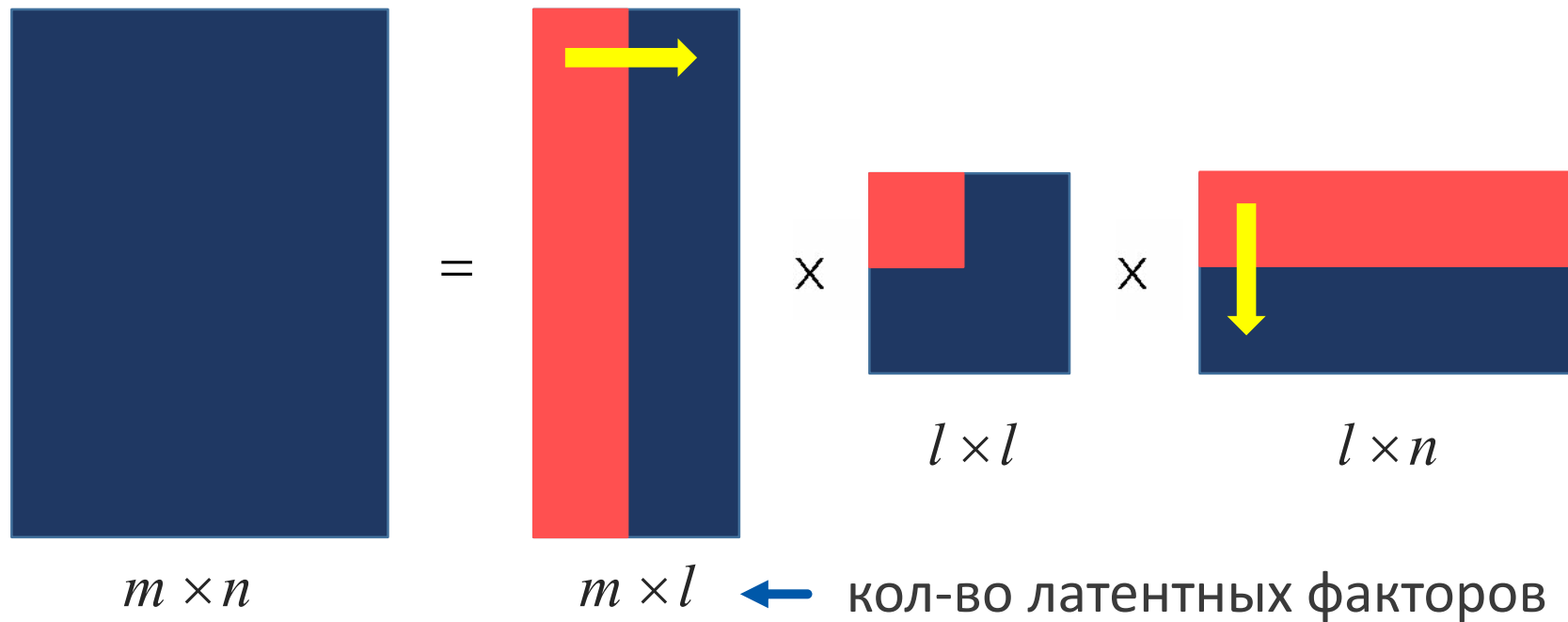
	4	5	6	7	8	9
						
	2		2	4	5	
	5		4			1
			5		2	
		1		5		4
			4			2
	4	5		1		

SVD для плотной матрицы

$$R = U\Sigma V^T$$

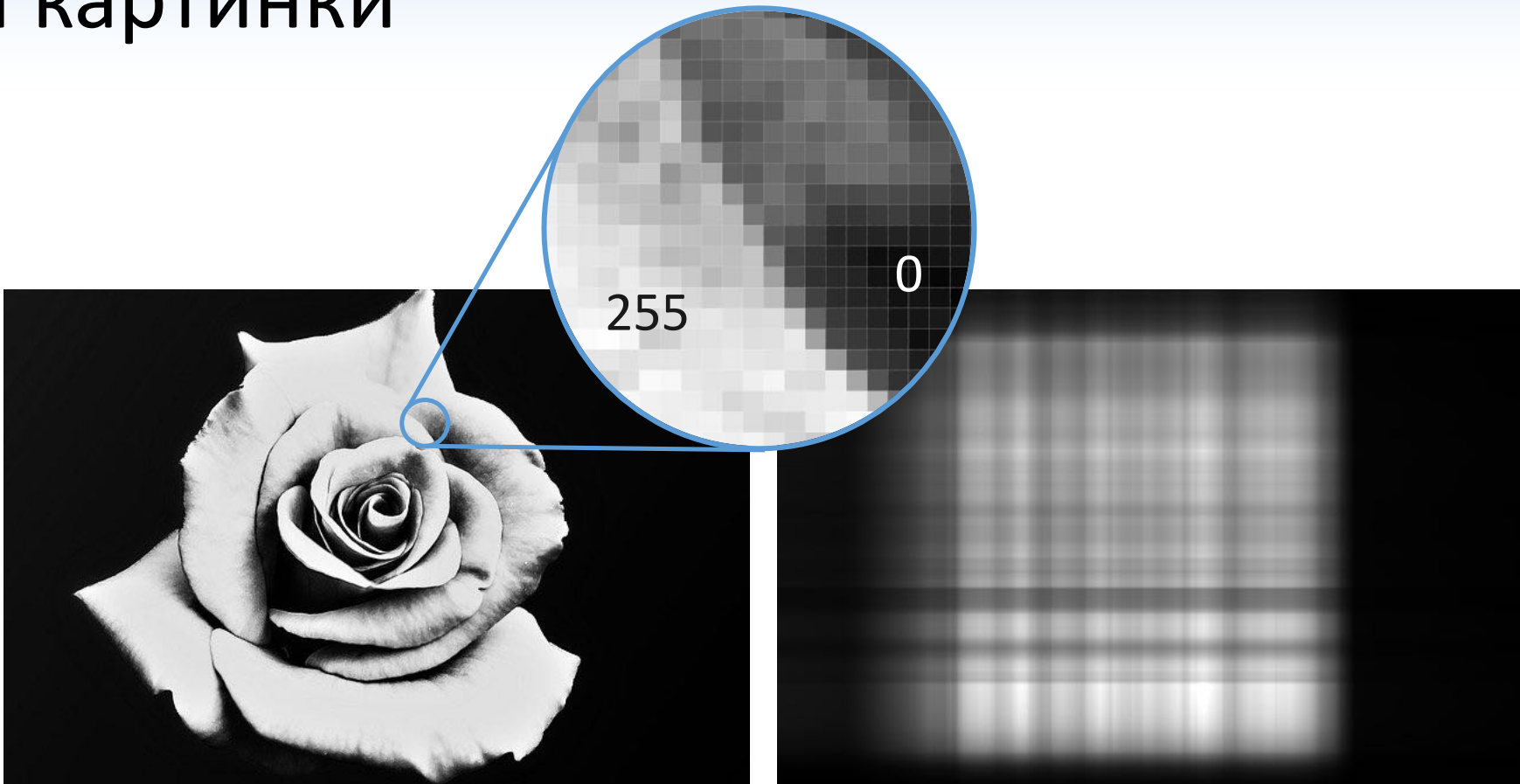
$$UU^T = U^T U = I \quad \leftarrow \text{единичная матрица}$$

$$VV^T = V^T V = I$$



Факторы отсортированы по вкладу в
приближение исходной матрицы

SVD для картинки



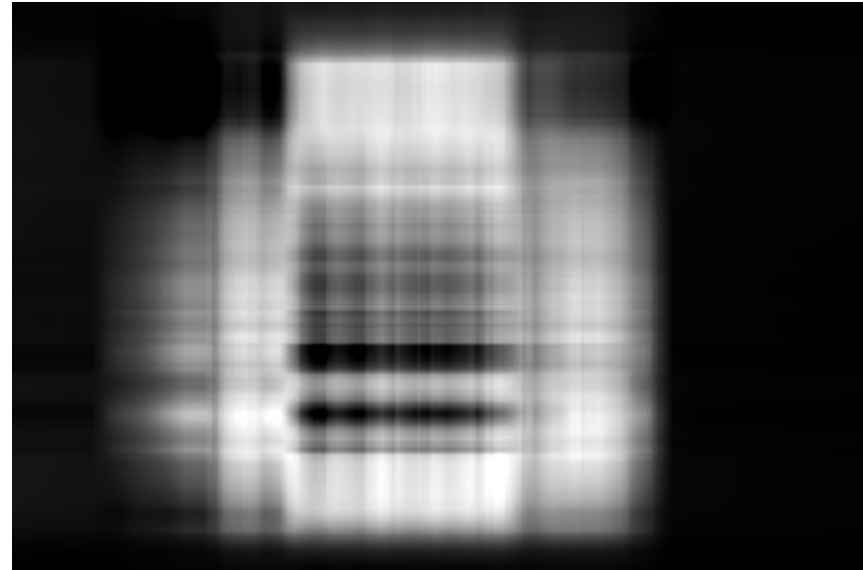
400 x 600 image
240000 values

400 x 1 / 1 x 1 / 1 x 600 matrices
1001 values

SVD для картинки



400 x 600 image
240000 values

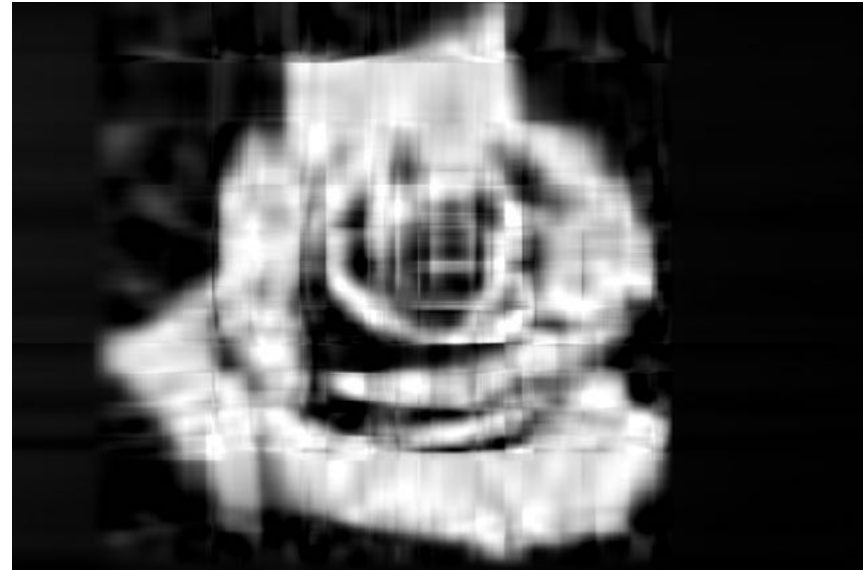


400 x 2 / 2 x 2 / 2 x 600 matrices
2004 values

SVD для картинки



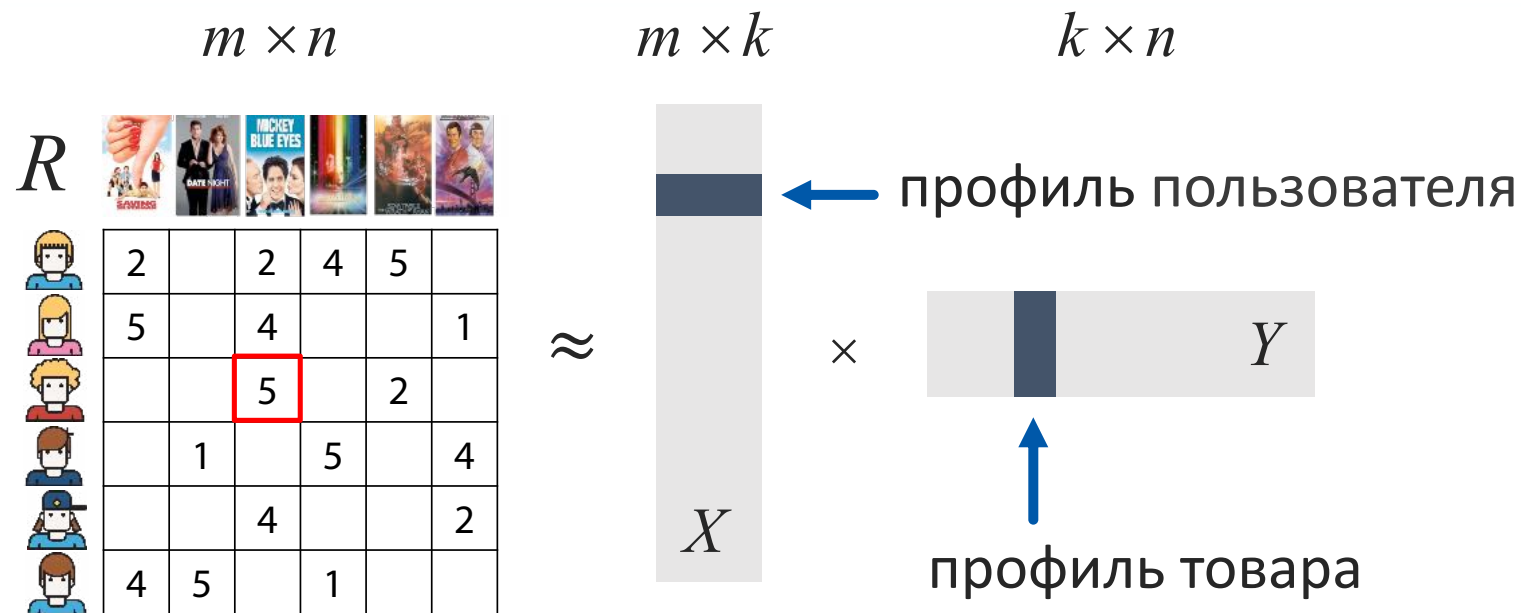
400 x 600 image
240000 values



400 x 9 / 9 x 9 / 9 x 600 matrices
9081 values

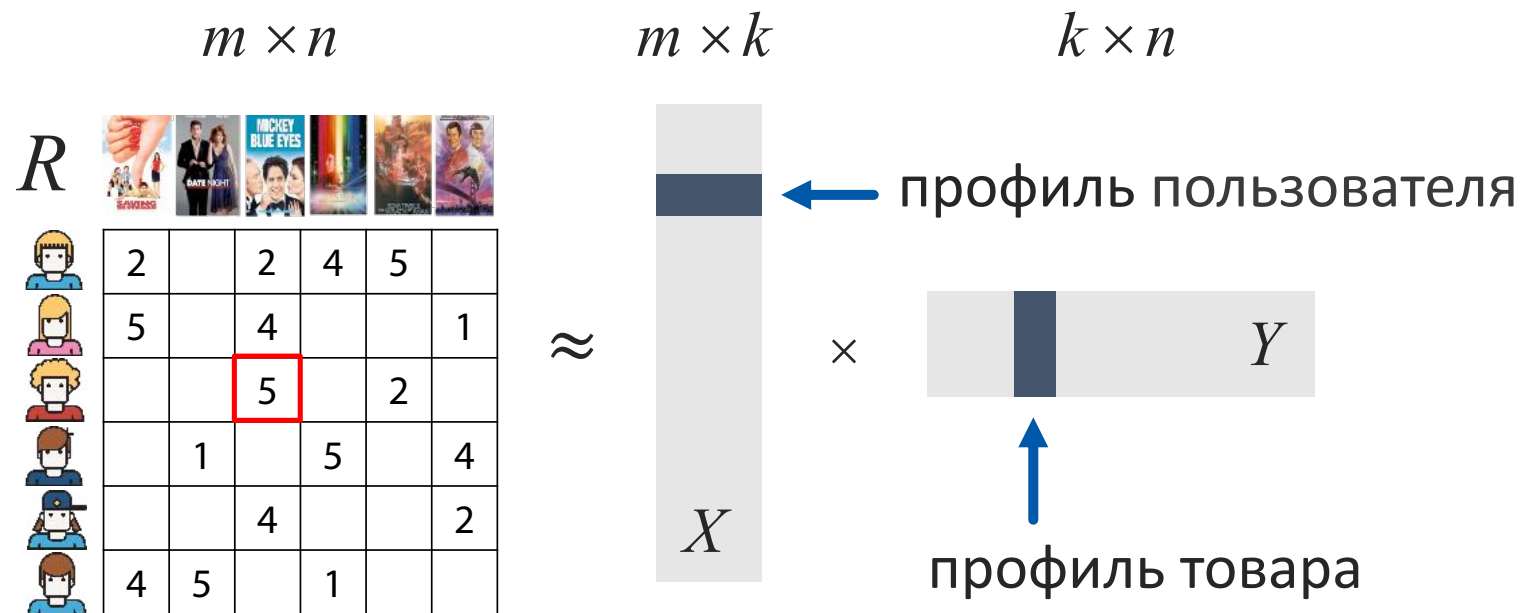
26x сжатие!

Funk SVD для разреженной матрицы



- Пользователи и товары погружаются в новое пространство (малой размерности k).
- Координаты в этом пространстве – «характеристики» товаров и пользователей.
- Похожесть в этом пространстве – это скалярное произведение!

Funk SVD для разреженной матрицы



Оптимизируем при помощи SGD:

$$\min_{X,Y} \sum_{(u,i) \in R} (r_{ui} - x_u^T y_i)^2 + \lambda (\|x_u\|^2 + \|y_i\|^2)$$

Интерпретация профилей товаров



Можно оценить похожести товаров

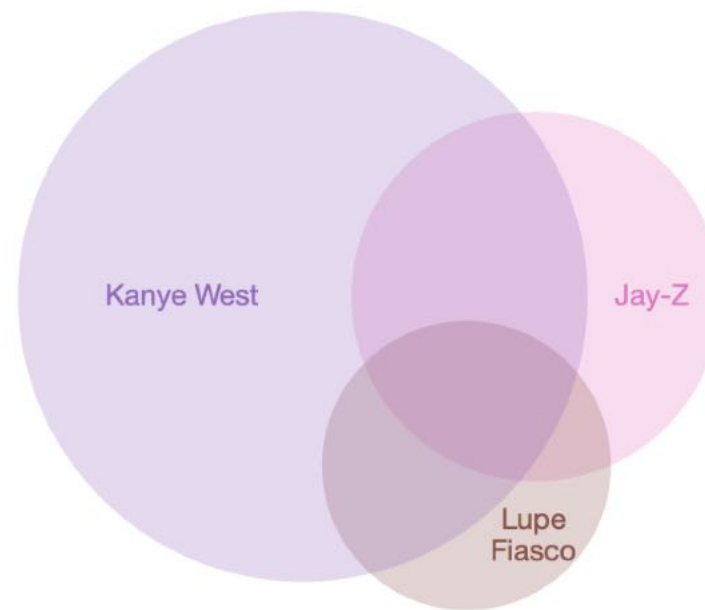


Пример похожестей товаров

Самые похожие на Kanye West
по косинусу между профилями исполнителей:

Artist	Implicit ALS
Lupe Fiasco	0.950
Jay-Z	0.928
T.I.	0.913
Lil Wayne	0.895
N*E*R*D	0.891

[more ...](#)



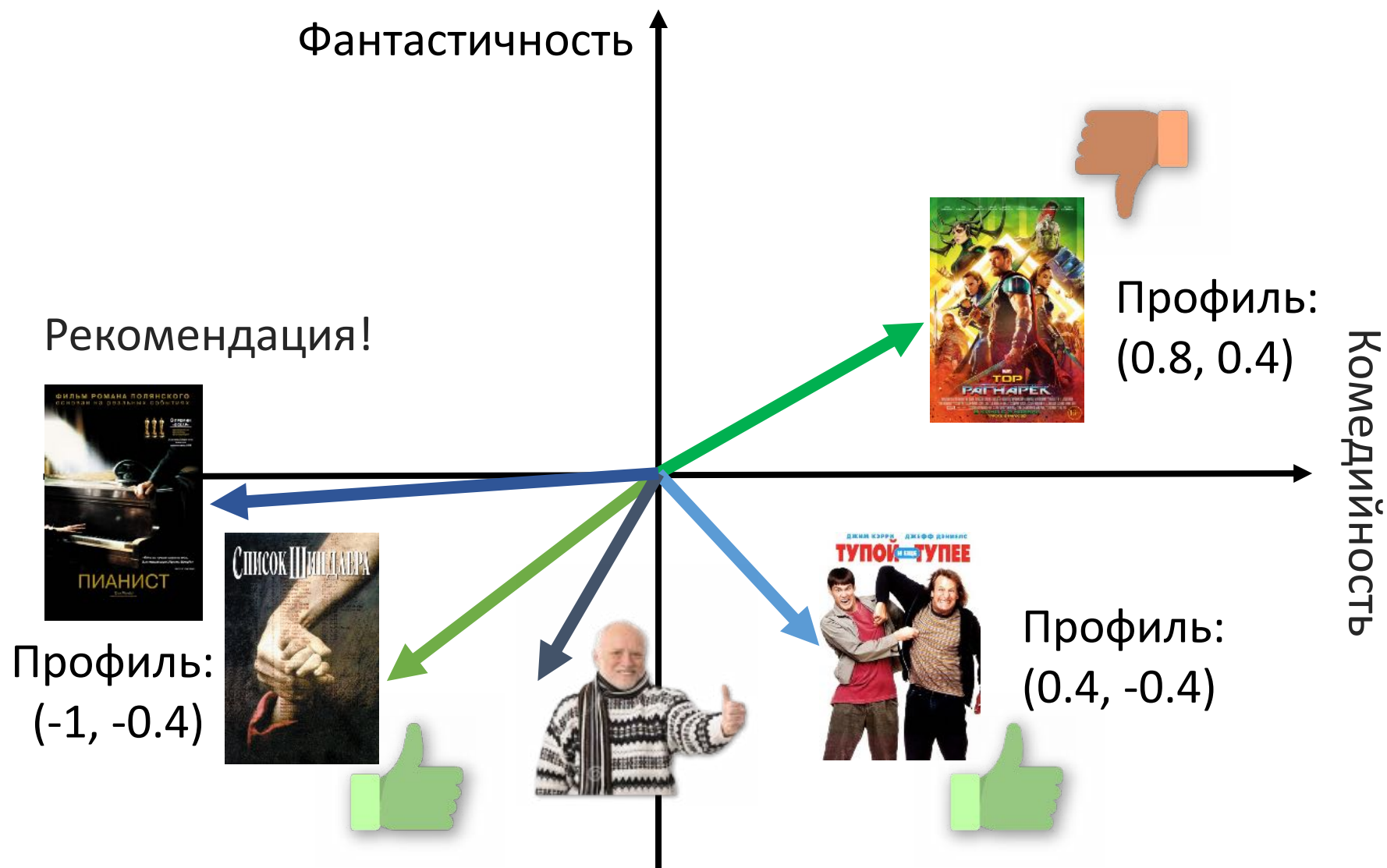
Профиль для нового пользователя



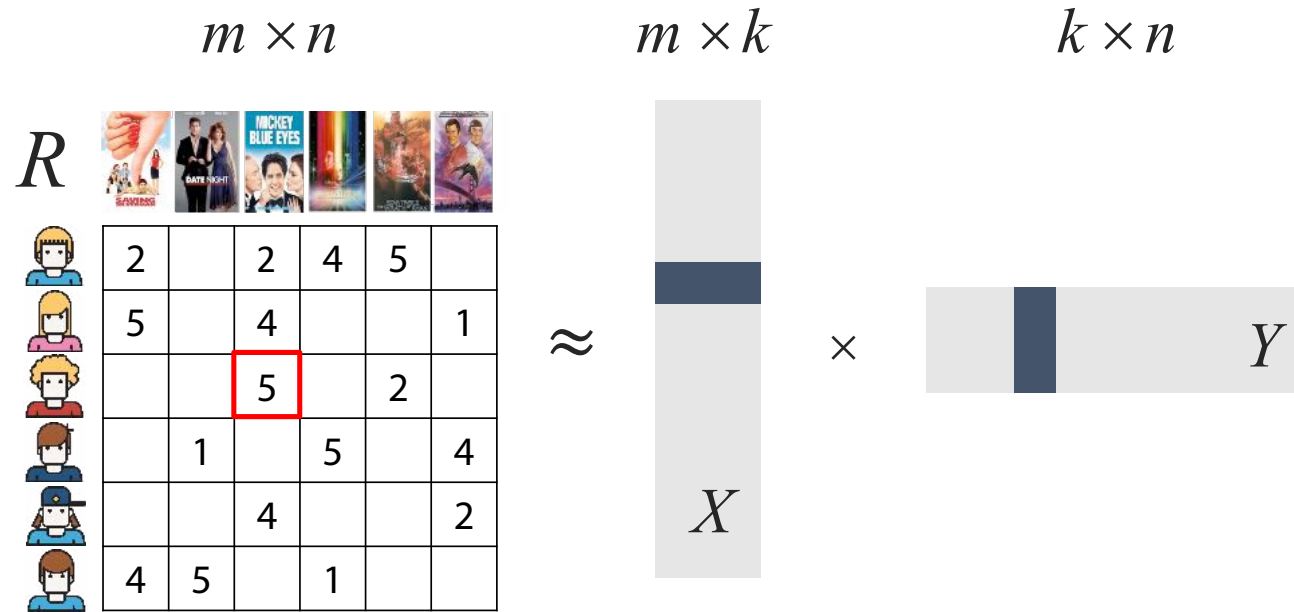
Профиль для нового пользователя



Рекомендация для пользователя



Alternating Least Squares (ALS)



$$\min_{X,Y} \sum_{(u,i) \in R} (r_{ui} - x_u^T y_i)^2 + \lambda (\|x_u\|^2 + \|y_i\|^2)$$




Alternating Least Squares (ALS)

$$\min_{X,Y} \sum_{(u,i) \in R} (r_{ui} - x_u^T y_i)^2 + \lambda (\|x_u\|^2 + \|y_i\|^2)$$

Инициализируем X и Y случайными значениями

В цикле:


- Фиксируем матрицу X (пользователей)
- Находим оптимальную матрицу Y (решаем гребневую регрессию для каждого товара)

	User 1 profile	×	≈	5
	User 2 profile			1
	User 3 profile			4

- И наоборот

Implicit ALS (iALS) для неявных оценок

как долго читал статью

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \text{ (по умолчанию)} \end{cases}$$


Понравилось?

$$c_{ui} = 1 + \alpha r_{ui}$$

Если не читал,
то уверенность
в $p_{ui} = 0$ маленькая

$$\min \sum_{u,i} c_{ui} (p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

Взвешенные потери




сумма по всем ячейкам (пропусков нет)

iALS: считается так же быстро

Решение регрессии

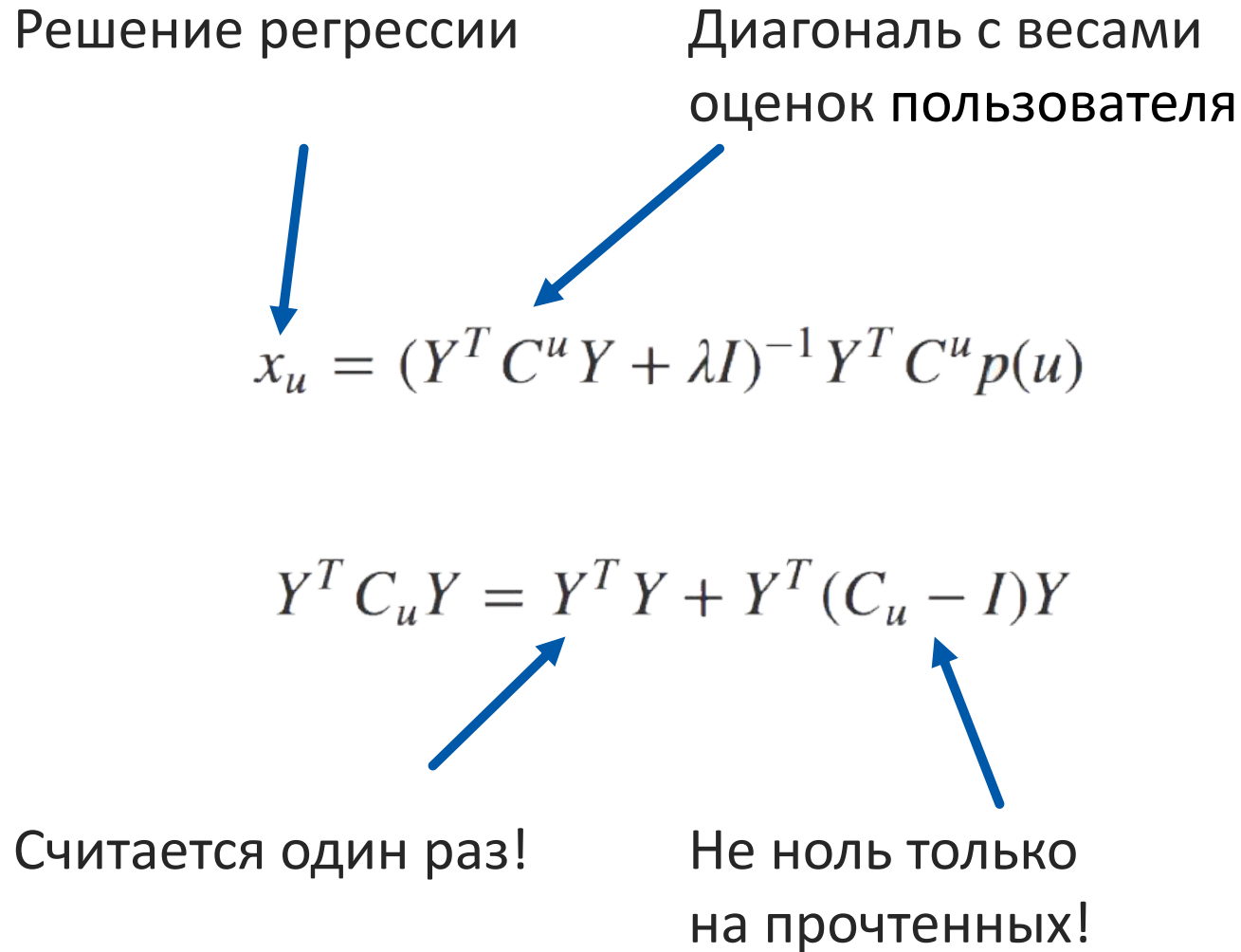
Диагональ с весами
оценок пользователя


$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$$

iALS: считается так же быстро

Решение регрессии

Диагональ с весами
оценок пользователя



The diagram illustrates the computational complexity of the iALS regression equation. It features two main equations. The top equation, $x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$, is annotated with two arrows: one from 'Решение регрессии' pointing to the inverse term, and another from 'Диагональ с весами оценок пользователя' pointing to the C^u term. The bottom equation, $Y^T C_u Y = Y^T Y + Y^T (C_u - I) Y$, is annotated with two arrows: one from 'Считается один раз!' pointing to the $Y^T Y$ term, and another from 'Не ноль только на прочтенных!' pointing to the $(C_u - I)$ term.

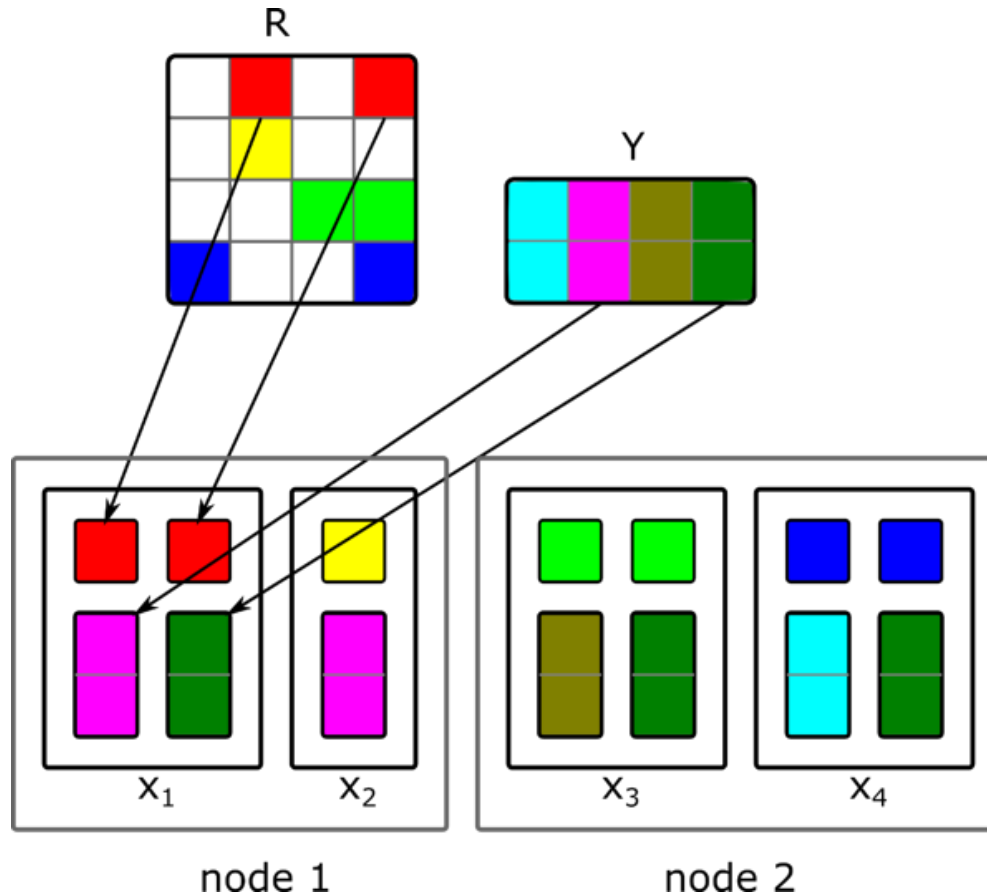
$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$$

$$Y^T C_u Y = Y^T Y + Y^T (C_u - I) Y$$

Считается один раз!

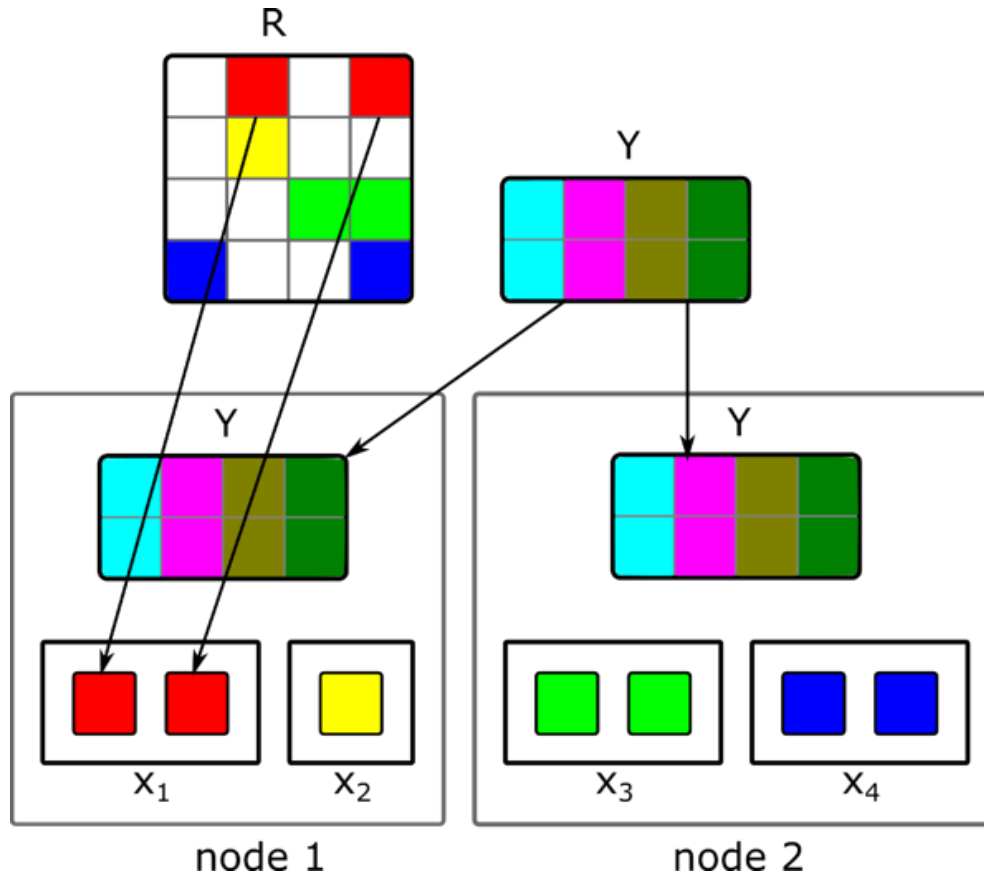
Не ноль только
на прочтенных!

Наивный параллельный ALS

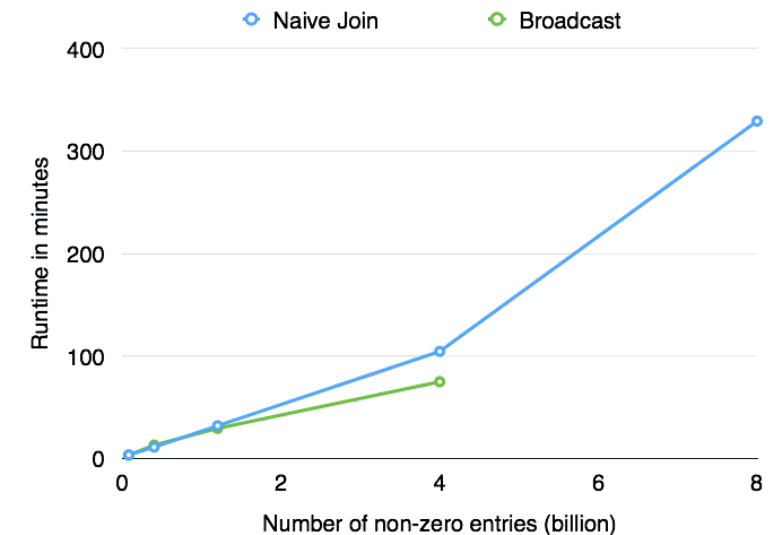


- $R = XY$, обновляем X
- Join R и Y по товарам $\rightarrow (u, r_{ui}, y_i)$
- Группируем по u
- На каждой машине пересчитываем x_u
- Один и тот же вектор y_i может понадобится нескольким пользователям на машине, и будет отправлен несколько раз

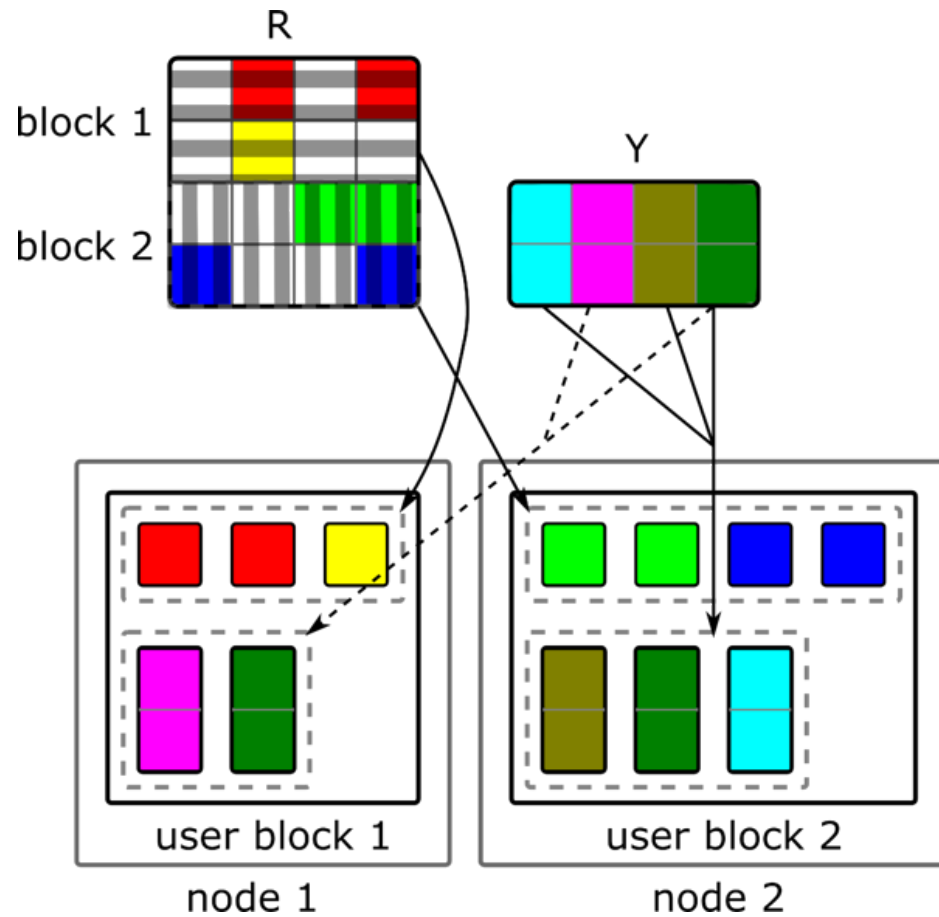
Реализация с broadcast Y



- Быстрее
- Имеет предел масштабирования (Y должен влезать в память)



Блочный ALS (как в Spark ML)



- Пользователи бьются на блоки, каждый блок обрабатывается на своей машине
- Один раз вычисляем какие товары понадобятся в каждом блоке (мапинг хранится в памяти)
- Пересылаются только нужные части матриц в память машин

Блочный ALS (как в Spark ML)

- Числом блоков можно управлять (параметр numBlocks)
- Чем их больше, тем выше шанс, что подматрица товаров для каждого блока влезет в память

```
from pyspark.ml.recommendation import ALS
```

```
als = ALS(maxIter=5, regParam=0.01,  
userCol="userId", itemCol="itemId",  
ratingCol="rating", numBlocks=10)
```

```
model = als.fit(training)
```

Ссылки

- <https://stanford.edu/~rezab/sparkworkshop/slides/xiangrui.pdf>
- <https://www.kaggle.com/tkm2261/outbrain-click-prediction/bigquery-is-cool-lb-0-63692>
- <http://cs229.stanford.edu/proj2016/report/JaiswalGopinathLimaye-OutbrainClickPrediction-report.pdf>
- <https://data-artisans.com/how-to-factorize-a-700-gb-matrix-with-apache-flink/>