Матричные разложения

Авторы:
Гринберг Петр
Пантелеев Даниил
Тряпицын Александр

QR-разложение

QR-разложение

- QR-разложение

$$A = QR,\ A,\ Q \in \mathbb{C}^{m\times m},\ R \in \mathbb{C}^{m\times n},\ Q^*Q = I,\ R :\uparrow \triangle$$

QR-разложение

- QR-разложение

$$A = QR,\ A,\ Q \in \mathbb{C}^{m\times m},\ R \in \mathbb{C}^{m\times n},\ Q^*Q = I,\ R :\uparrow \triangle$$

- Thin QR:

$$A = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_1 \\ 0 \end{pmatrix} = Q_1 R_1$$

QR-разложение

- QR-разложение

$$A = QR,\ A,\ Q \in \mathbb{C}^{m\times m},\ R \in \mathbb{C}^{m\times n},\ Q^*Q = I,\ R :\uparrow \triangle$$

- Thin QR:

$$A = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_1 \\ 0 \end{pmatrix} = Q_1 R_1$$

```
import numpy as np
np.linalg.qr(A)
```

QR-разложение: как считать?

QR-разложение: как считать?

- Грамм-Шмидт (долго)

QR-разложение: как считать?

- Грамм-Шмидт (долго)
- Матрицы Хаусхолдера: $H(v) = 1 - vv^*, \|v\|_2 = 1$

  $\forall a, b \in \mathbb{C}^n, \|a\|_2 = \|b\|_2, \exists \gamma \in \mathbb{C} : |\gamma| = 1, v \in \mathbb{C}^n : \|v\|_2 = 1 \Rightarrow H(v)a = \gamma b$

  Зануляем элементы под главной диагональю:

$$H_1 H_2 ... H_n A = R$$

QR-разложение: зачем надо?

QR-разложение: зачем надо?

- Ортопроекторы — не нужно считать обратные:

$$A(A^T A)^{-1} A^T = QR(R^T Q^T QR)^{-1} R^T Q^T = QRR^{-1} R^{-T} R^T Q^T = QQ^T$$

QR-разложение: зачем надо?

- Ортопроекторы — не нужно считать обратные:

$$A(A^TA)^{-1}A^T = QR(R^TQ^TQR)^{-1}R^TQ^T = QRR^{-1}R^{-T}R^TQ^T = QQ^T$$

- Используется в ALS, в алгоритмах для поиска SVD, NMF и др.

SVD-разложение

SVD-разложение

- SVD-разложение

$$A = USV^*, U^*U = I = V^*V, S = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$$

$$\sigma_1 \geq \ldots \geq \sigma_r > 0, \sigma_{r+1} = \ldots = \sigma_n = 0$$

SVD-разложение

- SVD-разложение

$$A = USV^*, U^*U = I = V^*V, S = \mathrm{diag}(\sigma_1, \dots, \sigma_n)$$

$$\sigma_1 \geq \dots \geq \sigma_r > 0, \sigma_{r+1} = \dots = \sigma_n = 0$$

- Thin SVD: обрезается до $k = \min(m, n)$

SVD-разложение

- SVD-разложение

$$A = USV^*, U^*U = I = V^*V, S = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$$

$$\sigma_1 \geq \ldots \geq \sigma_r > 0, \sigma_{r+1} = \ldots = \sigma_n = 0$$

- Thin SVD: обрезается до $k = \min(m, n)$
- Compact SVD: обрезается до $r = \mathrm{rk}(A)$

SVD-разложение

- SVD-разложение

$$A = USV^*, U^*U = I = V^*V, S = \mathrm{diag}(\sigma_1, ..., \sigma_n)$$

$$\sigma_1 \geq ... \geq \sigma_r > 0, \sigma_{r+1} = ... = \sigma_n = 0$$

- Thin SVD: обрезается до $k = \min(m, n)$
- Compact SVD: обрезается до $r = \mathrm{rk}(A)$
- Truncated SVD: обрезается до заданного t

SVD-разложение

- SVD-разложение

$$A = USV^*, U^*U = I = V^*V, S = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$$

$$\sigma_1 \geq \ldots \geq \sigma_r > 0, \sigma_{r+1} = \ldots = \sigma_n = 0$$

- Thin SVD: обрезается до $k = \min(m, n)$
- Compact SVD: обрезается до $r = \operatorname{rk}(A)$
- Truncated SVD: обрезается до заданного t

```
np.linalg.svd(A, full_matrices=True) # False for thin SVD
```

SVD-разложение: важное свойство

SVD-разложение: важное свойство

Теорема (Эккарт-Янг):

Пусть $k < \mathrm{rk}(A) = r$ и A_k - truncated SVD с $t = k$. Тогда:

$$\min_{B:\ \mathrm{rk(B)} \leq k} \|A - B\| = \|A - A_k\|$$

Где $\|\|$ - любая инвариантная норма.

SVD-разложение: как считать?

SVD-разложение: как считать?

Столбцы V и числа σ_i^2 являются соб. векторами и соб. знач. $A^T A$

Golub, G. ; Kahan, W.: Calculating the Singular Values and Pseudo-Inverse of a Matrix (1965)

SVD-разложение: как считать?

Столбцы V и числа σ_i^2 являются соб. векторами и соб. знач. $A^T A$

1. Задача сводится к поиску с.в. и с.з. у $A^T A$, AA^T, $\begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix}$

Golub, G. ; Kahan, W.: Calculating the Singular Values and Pseudo-Inverse of a Matrix (1965)

SVD-разложение: как считать?

Столбцы V и числа σ_i^2 являются соб. векторами и соб. знач. A^TA

1. Задача сводится к поиску с.в. и с.з. у A^TA, AA^T, $\begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix}$
2. Приводим A к бидиагональной (B)

$$PAQ = B, \text{где } P = H_1, ...H_k, Q = H'_1, ..., H'_{k'}$$

Golub, G. ; Kahan, W.: Calculating the Singular Values and Pseudo-Inverse of a Matrix (1965)

SVD-разложение: как считать?

Столбцы V и числа σ_i^2 являются соб. векторами и соб. знач. $A^T A$

1. Задача сводится к поиску с.в. и с.з. у $A^T A$, AA^T, $\begin{pmatrix} 0 & A^T \\ A & 0 \end{pmatrix}$
2. Приводим A к бидиагональной (B)

$$PAQ = B, \text{где } P = H_1, ...H_k, Q = H'_1, ..., H'_{k'}$$

3. QR-алгоритм для $T = C$, где C - одна из матриц в п.1 с B

Golub, G. ; Kahan, W.: Calculating the Singular Values and Pseudo-Inverse of a Matrix (1965)

NMF (Non-negative matrix factorization)

NMF (Non-negative matrix factorization)

Пусть $V \in \mathbb{R}^{m \times n}$ такова, что $v_{ij} \geq 0$. Тогда NMF это W, H:

$$W, H = \underset{W \in \mathbb{R}^{m \times r}, H \in \mathbb{R}^{r \times n}}{\operatorname{argmin}} \|V - WH\| \text{ где } w_{ij} \geq 0, h_{ij} \geq 0$$

Выбор нормы зависит от постановки задачи. r - желаемый ранг

NMF (Non-negative matrix factorization)

Пусть $V \in \mathbb{R}^{m \times n}$ такова, что $v_{ij} \geq 0$. Тогда NMF это W, H:

$$W, H = \underset{W \in \mathbb{R}^{m \times r}, H \in \mathbb{R}^{r \times n}}{\operatorname{argmin}} \|V - WH\| \text{ где } w_{ij} \geq 0, h_{ij} \geq 0$$

Выбор нормы зависит от постановки задачи. r - желаемый ранг

```
from sklearn.decomposition import NMF
model = NMF(n_components=r)
W = model.fit_transform(V)
H = model.components_
```

NMF: как считать?

NMF: как считать?

- Неитерационный алгоритм: NP-полная задача

NMF: как считать?

- Неитерационный алгоритм: NP-полная задача
- Итерационный алгоритм:

$$W_0, H_0 = \mathrm{initWH}(), \qquad W_{k+1} = f(W_k, H_k, V), \qquad H_{k+1} = f(W_{k+1}, H_k, V)$$

Обычно используются различные варианты ALS алгоритма

NMF: как считать?

- Неитерационный алгоритм: NP-полная задача
- Итерационный алгоритм:

$$W_0, H_0 = \mathrm{initWH}(), \qquad W_{k+1} = f(W_k, H_k, V), \qquad H_{k+1} = f(W_{k+1}, H_k, V)$$

Обычно используются различные варианты ALS алгоритма

Для инициализации:

1. Первые r столбцов V в W, H - случайная

NMF: как считать?

- Неитерационный алгоритм: NP-полная задача
- Итерационный алгоритм:

$$W_0, H_0 = \mathrm{initWH}(), \qquad W_{k+1} = f(W_k, H_k, V), \qquad H_{k+1} = f(W_{k+1}, H_k, V)$$

Обычно используются различные варианты ALS алгоритма

Для инициализации:

1. Первые r столбцов V в W, H - случайная
2. $g(A), g(B)$ для A, B из $V \sim ASB^T$ - truncated SVD с $t = r$

LU-разложение

LU-разложение

- LU-разложение:

$$A = LU$$

Где L - унинижнетреугольная, U - верхнетреугольная

LU-разложение

- LU-разложение:

$$A = LU$$

  Где L - унинижнетреугольная, U - верхнетреугольная
- LDL: $A = LDL^*$ для $A = A^*$

LU-разложение

- LU-разложение:

$$A = LU$$

  Где L - унинижнетреугольная, U - верхнетреугольная
- LDL: $A = LDL^*$ для $A = A^*$
- Холецкого: $A = LL^*, L$ - нижнетреугольная для $A = A^* > 0$

LU-разложение

- LU-разложение:

$$A = LU$$

  Где L - унинижнетреугольная, U - верхнетреугольная

- LDL: $A = LDL^*$ для $A = A^*$
- Холецкого: $A = LL^*, L$ - нижнетреугольная для $A = A^* > 0$

```
from scipy.linalg import lu
P, L, U = lu(A)
```

LU-разложение: как считать и зачем?

LU-разложение: как считать и зачем?

- *Как считать*:
  По очереди Гаусс для каждого столбца

$$Z_n ... Z_1 A = U$$

LU-разложение: как считать и зачем?

- *Как считать*:
  По очереди Гаусс для каждого столбца

$$Z_n \ldots Z_1 A = U$$

- *Зачем*:
  Хотим решить $Ax = b \Leftrightarrow$ умножить b на A^{-1}

$$Ax = b \Leftrightarrow LUx = b$$

$$Ly = b, Ux = y$$

Применения в Машинном Обучении

PCA (Principal Component Analysis)

PCA (Principal Component Analysis)

Имеем:

набор данных большой размерности

PCA (Principal Component Analysis)

Имеем:

набор данных большой размерности

Хотим:

уменьшить число признаков, сохранив как можно больше информации

Зачем это нужно?

Зачем это нужно?

- Чтобы избежать переобучения

Зачем это нужно?

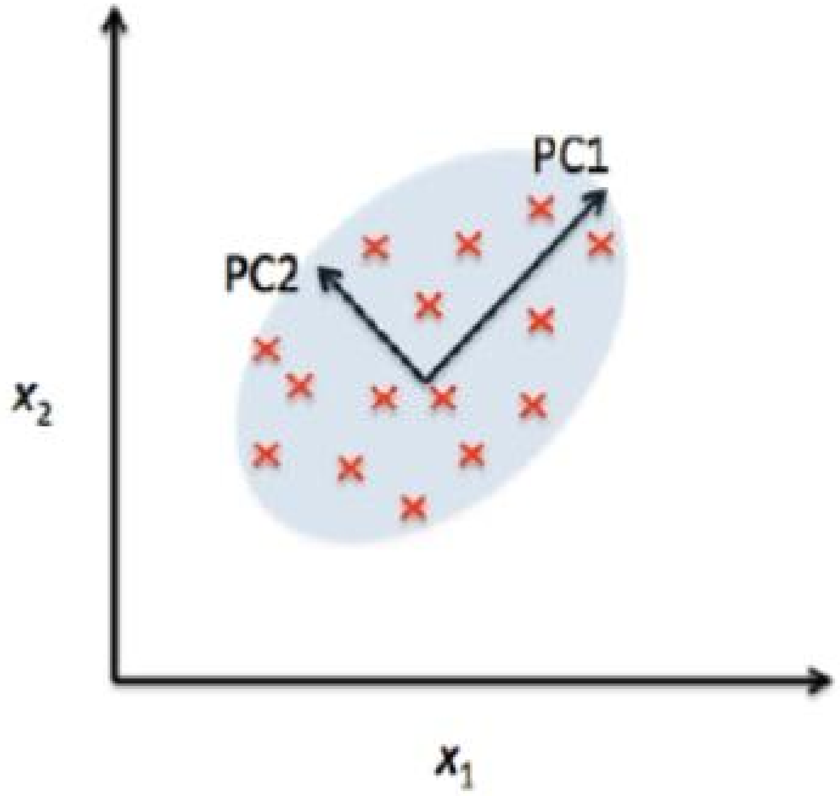
- Чтобы избежать переобучения
- Можно сильно уменьшить размер матрицы данных

Зачем это нужно?

- Чтобы избежать переобучения
- Можно сильно уменьшить размер матрицы данных
- В некоторых задачах помогает избавиться от шумов

Основная идея:

Перенаправим оси признаков так, чтобы их разброс был виднее



Шаг 1: Стандартизация

1. Вычитаем из каждого признака среднее по нему

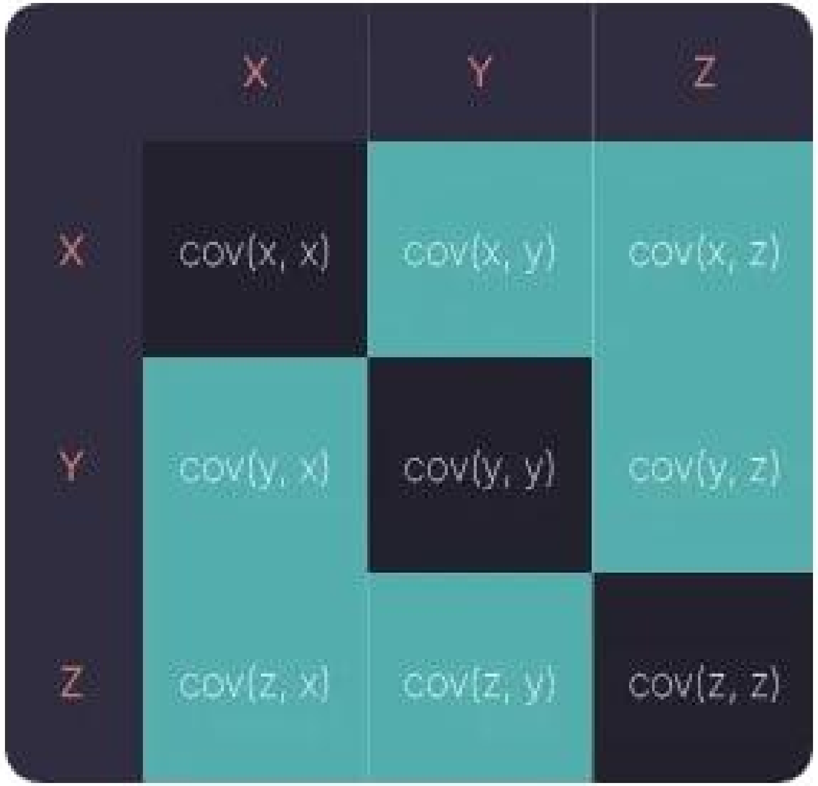
2. Делим каждый признак на RMSD

$$\text{RMSD} = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}} = \sqrt{MSE(x)}$$

Шаг 2: Матрица ковариаций

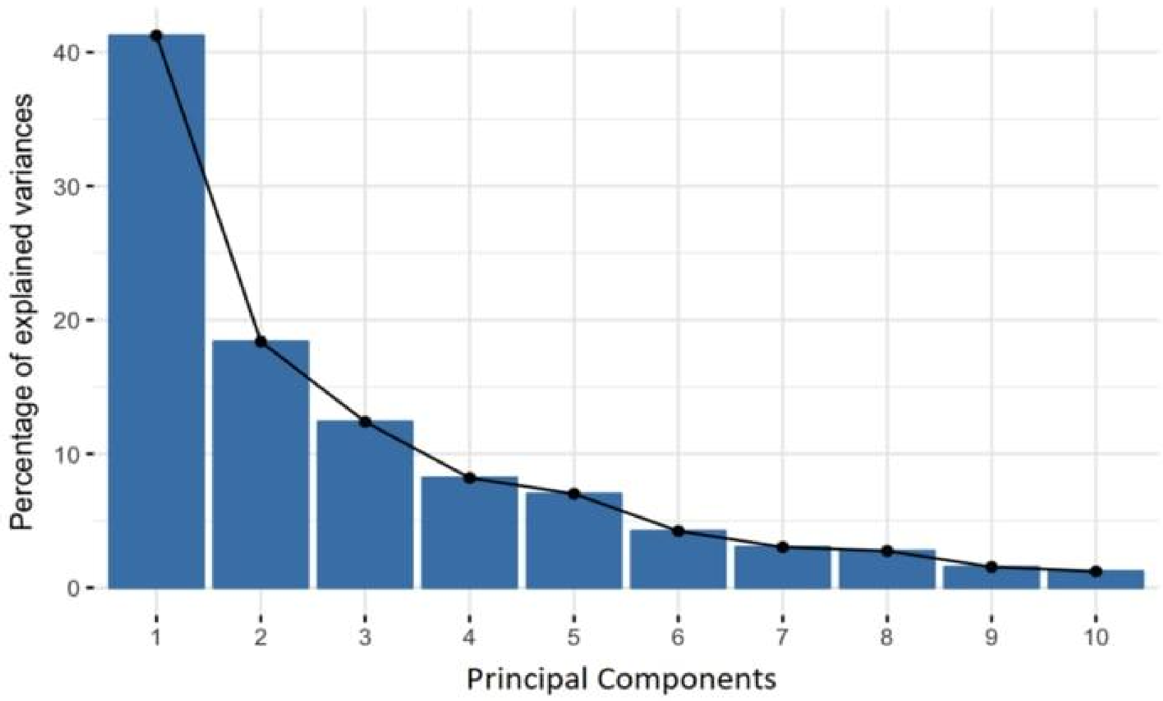
$$var(x) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}$$

$$cov(x, y) = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{n}$$



Шаг 3: Собственные векторы

1. Считаем собственные векторы матрицы ковариаций
2. Сортируем их по убыванию собственных значений



Шаг 4: Матрица признаков

1. Выбираем t главных компонент, которые дают нужное приближение

2. Записываем их по столбцам, получаем матрицу признаков

Шаг 5: Результат

A - Стандартизированная исходная матрица данных

F - Матрица признаков

A' - Новая матрица данных

$$\underbrace{\mathrm{A'}}_{m\times t} = \underbrace{\mathrm{A}}_{m\times n} \times \underbrace{\mathrm{F}}_{n\times t}$$

При чем тут матричные разложения?

При чем тут матричные разложения?

Ответ: при поиске главных компонент ищется SVD от матрицы данных

C - матрица ковариации

$$A = U \times S \times V^T$$

В матрице S^2 собственные значения C
В матрице V^T собственные векторы C

PCA (Principal Component Analysis)

Не стоит использовать, если:

1. В данных почти нет корреляции
2. Есть классы, зависимые от каких-то признаков
3. Нормализация данных невозможна
4. Важна интерпретируемость

В остальных случаях:

```
from sklearn.decomposition import PCA
```

LSA (Latent Semantic Analysis)

LSA (Latent Semantic Analysis)

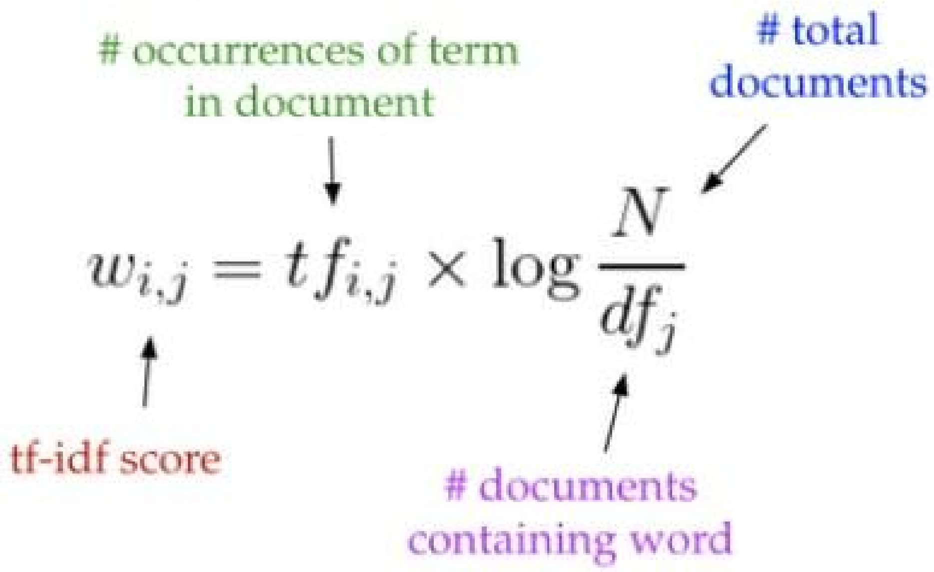
Имеем:

матрица term-document

term-document

- Обычно слова на тексты
- Внутри ячеек частоты использования в тексте
- Часто в ячейках tf-idf

tf-idf



Большие веса имеют термы, часто встречающиеся в одном тексте, но редко среди всех текстов

LSA (Latent Semantic Analysis)

Имеем:

матрица term-document

Хотим:

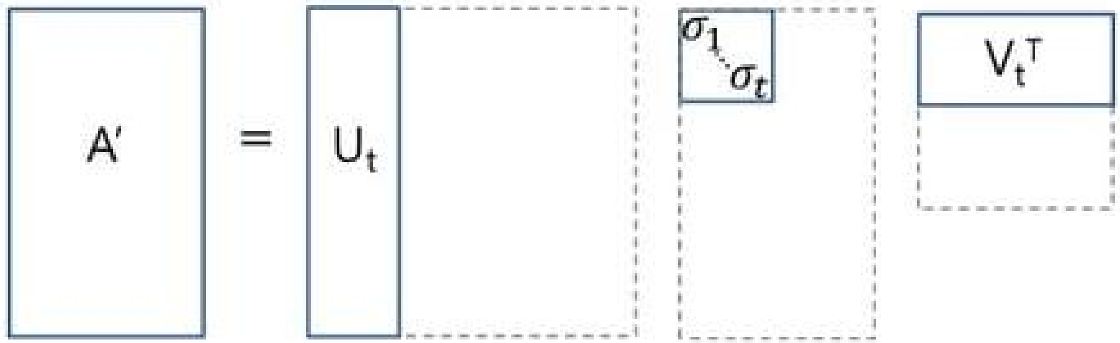
избавиться от шумов, разреженности

Зачем это нужно?

- Считать зависимости для пар term-term
- Считать зависимости для пар term-document
- Считать зависимости для пар document-document

Основная идея:

Найдем SVD и обрежем часть сингулярных чисел в матрице S, чтобы убрать часть зависимостей



$$A \approx U_t \times S_t \times V_t^T$$

Какое взять t?

- Подбирается эмпирически
- Для маленьких A обычно 5-10%
- Для больших A до 1-2%, иногда меньше

LSA (Latent Semantic Analysis)

По сути весь алгоритм заключается в нахождении Truncated SVD

```
from sklearn.decomposition import TruncatedSVD
```

На практике обычно используются 2 дополнения этого алгоритма:

- pLSA (Probabilistic LSA) - вероятностный подход к LSA
- LDA (Latent Dirichlet Allocation) - тот же вероятностный подход, но на основе распределения Дирихле

NMF in Topic Modelling

Имеем:

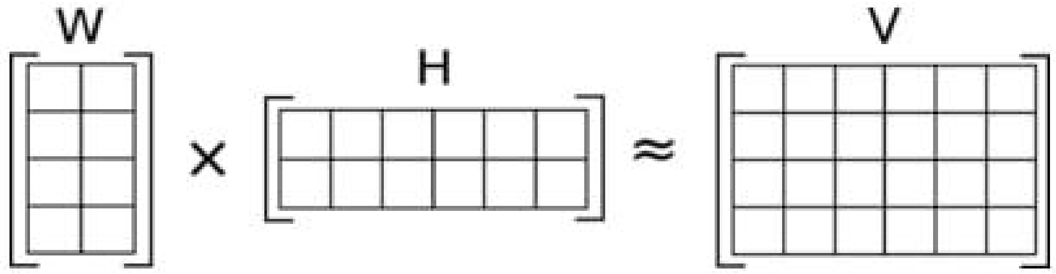
матрица term-document

Хотим:

Уменьшить веса менее значимых слов в наборе данных

Основная идея:

Посчитать NMF-разложение



В матрице H эмбеддинги, в матрице W семантические связи термов с текстами

W: распределение текстов (вертикаль) по темам (горизонталь)
H: распределение тем (вертикаль) по словам (горизонталь)

NMF in Topic Modelling

Весь алгоритм по сути это нахождение NMF-разложения

```
from sklearn.decomposition import NMF
```

Есть 2 вида встроенных алгоритмов оптимизации:

- Coordinate Decent Solver ('cd')
- Multiplicative Update Solver ('mu')

ВОПРОСЫ?

Матричные разложения

Рекомендательные системы

Источник части картиночек:
https://www.youtube.com/watch?v=ZspR5PZemcs

Постановка задачи

Дано:

- Множество пользователей
- Множество фильмов
- R_{mu} – Ratings – оценка фильма m пользователем u. В этой матрицы могут быть пропуски.

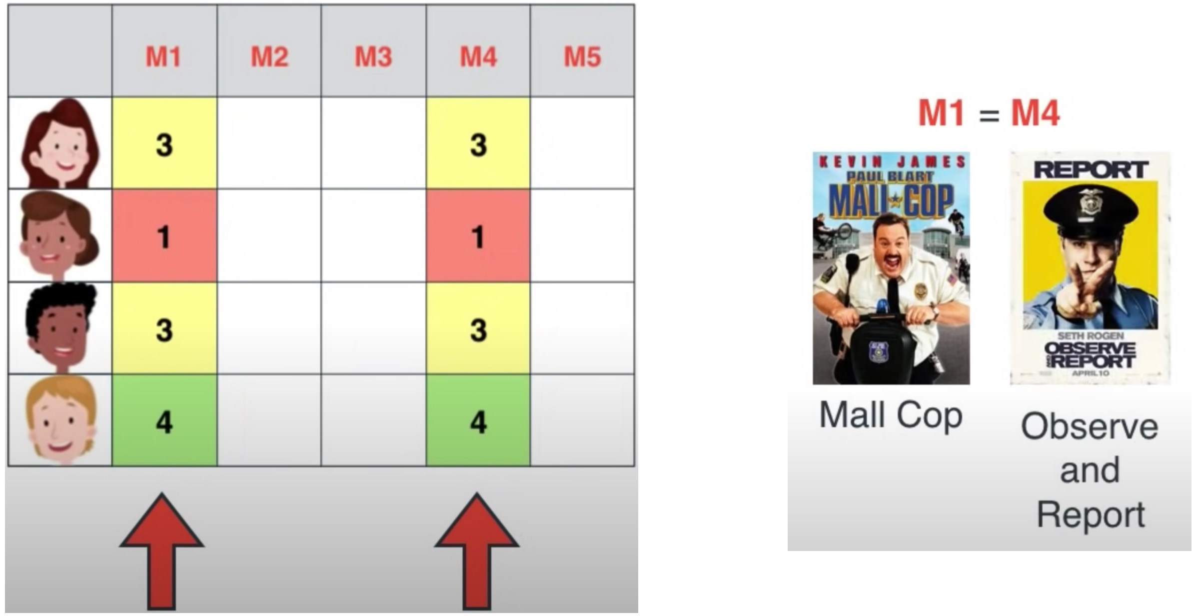
Хочется:

Для каждого пользователя в выдавать k наиболее интересных ему фильмов. По смыслу – заполнять пропуски в R.

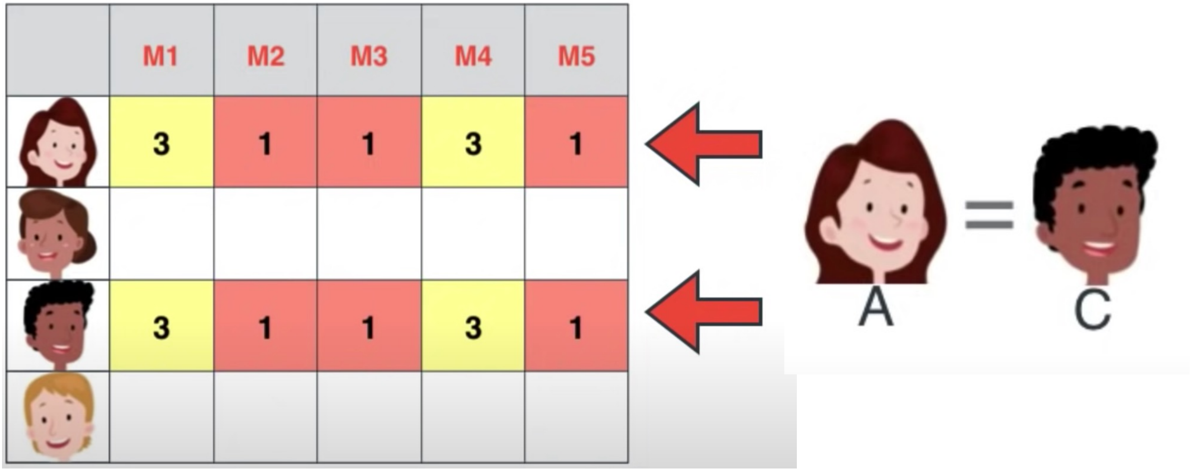
	M 1	M 2	M 3	M 4	M 5
	3	1	1	3	1
	1	2	4	1	3
	3	1	1	3	1
	4	3	5	4	4

Movie 1 Movie 2 Movie 3 Movie 4 Movie 5

Подходы к задаче



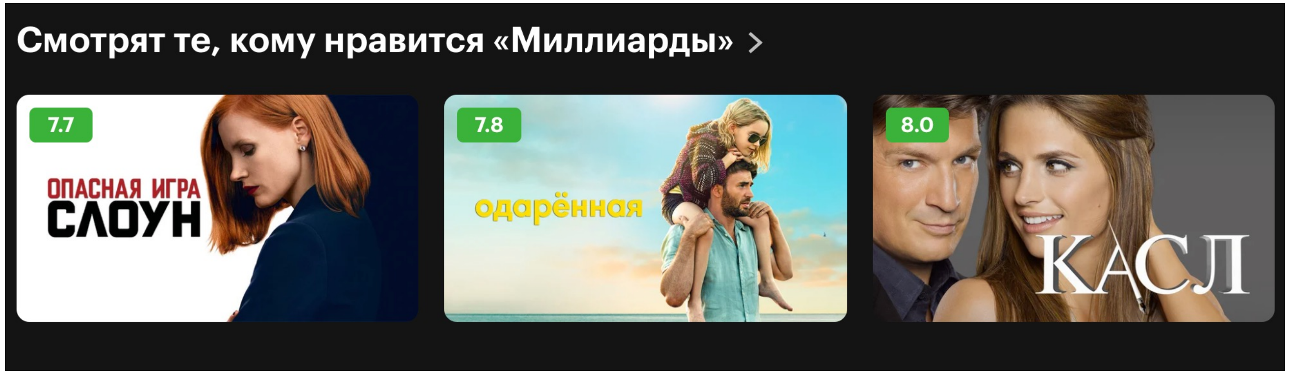
- Контентная фильтрация – рекомендуем пользователю, основываясь на контенте фильмов

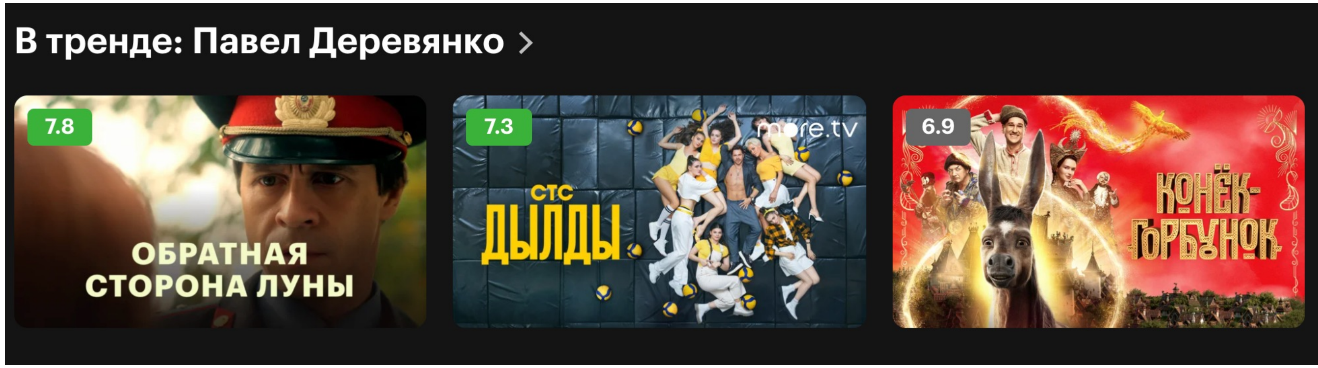


- Коллаборативная фильтрация – рекомендуем пользователю, основываясь на статистике

Наивная интуиция

1. Для каждого фильма находим k ближайших (наиболее похожих) фильмов
2. Для пользователя берем последний понравившийся ему фильм – X (тут может быть много разных эвристик)
3. Берем k ближайший к фильму X и называем рубрику «потому что вы смотрели X»





Можно делать и по актерам, если придумать как агрегировать оценку для актера по оценкам фильмов

Наивная реализация

- Для каждого фильма находим k ближайших фильмов по евклидовым расстояниям между соответствующими столбцами оценок:

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

- По косинусным расстояниям:

$$\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}}$$

- По коэффициенту корреляции Пирсона:

$$\frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2(y_i - \bar{y})^2}}$$

- Curse of dimensionality – при большом количестве пользователей (размерности пространства векторов оценок) – евклидовы расстояния распределены плотно

- Не учитываются различия в пользователях – есть оптимисты, есть пессимисты

- Не посчитаешь в рантайме – большое число пользователей. O(|M||U|)
- Для предпосчета требуется огромные вычислительные ресурсы. O(|M|²|U|)

Наивная реализация

Все еще остаются проблемы:

1. Вычислительная сложность:
   - рантайм - O(|M||U|)
   - предподсчет - $O(|M|^2|U|)$
2. Новые фильмы не будут иметь оценок, пока их не посмотрят много пользователей – рекомендации по этому фильму будут плохими, а сам фильм почти не будем рекомендовать
3. Популярные фильмы будем рекомендовать чаще (по сути противоположность предыдущему пункту)
4. Нет рекомендаций для новых пользователей
5. Выглядит хиленько, никаких связей фильмов друг с другом и пользователей друг с другой не учитываем

Наивная реализация еще

Есть еще ряд наивных реализаций со схожим списком проблем

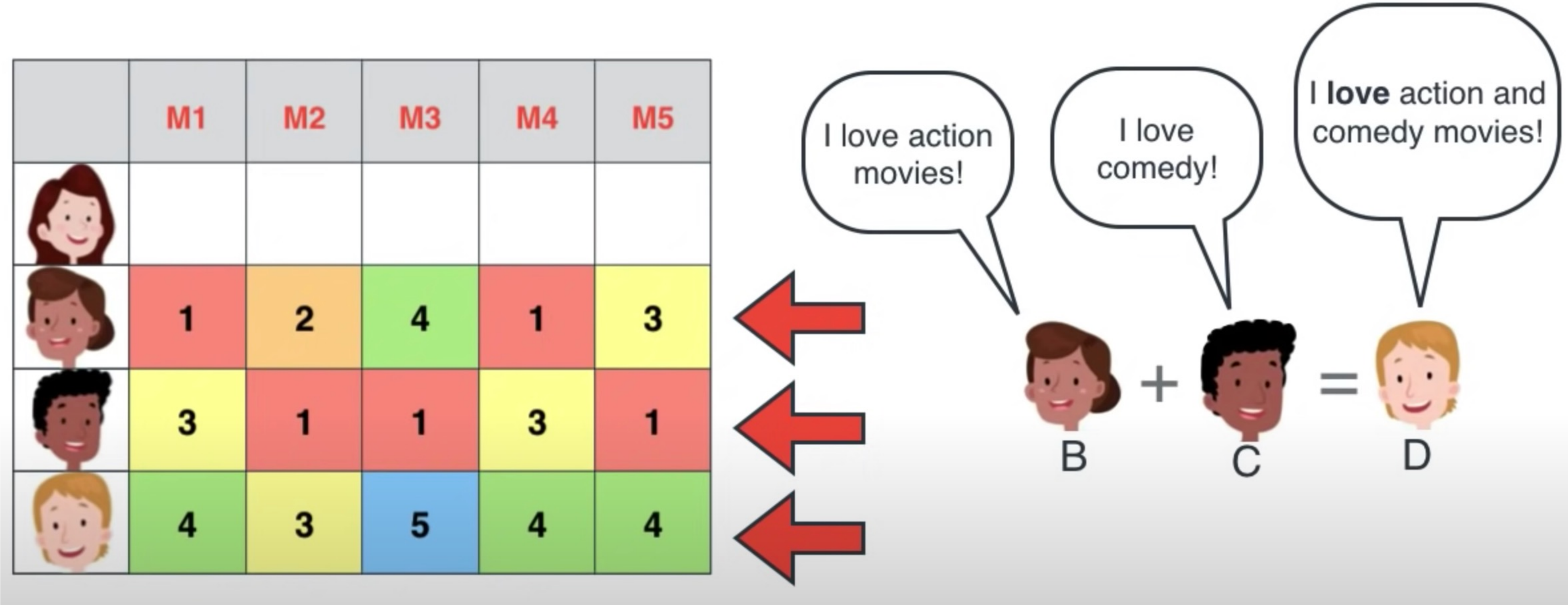
- Выбрать сначала сколько-то наиболее похожих пользователей, а потом взять топ по оценкам из их фильмов
- Выбрать несколько самых оцененных пользователем фильмов, а потом взять топ по оценкам из похожих на них фильмов
- Выбрать несколько самых оцененных пользователем фильмов, а потом взять топ по произведению оценок и похожести из похожих на них фильмов
- …

Матричные разложения - интуиция

- Матрица R наверняка разреженная – каждый пользователь посмотрел не слишком много фильмов
- Хорошо бы разложить R ≈ U @ M [Rating = User @ Movie], где ограничить ширину U (высоту M). Что делать с пропусками?
- U - по строкам пользователи, по столбцами - “свойство фильма” (комедия, драма, …), значение - насколько свойство нравится пользователю
- M - по строкам “свойства фильмов” , по столбцам - насколько свойство описывает фильм

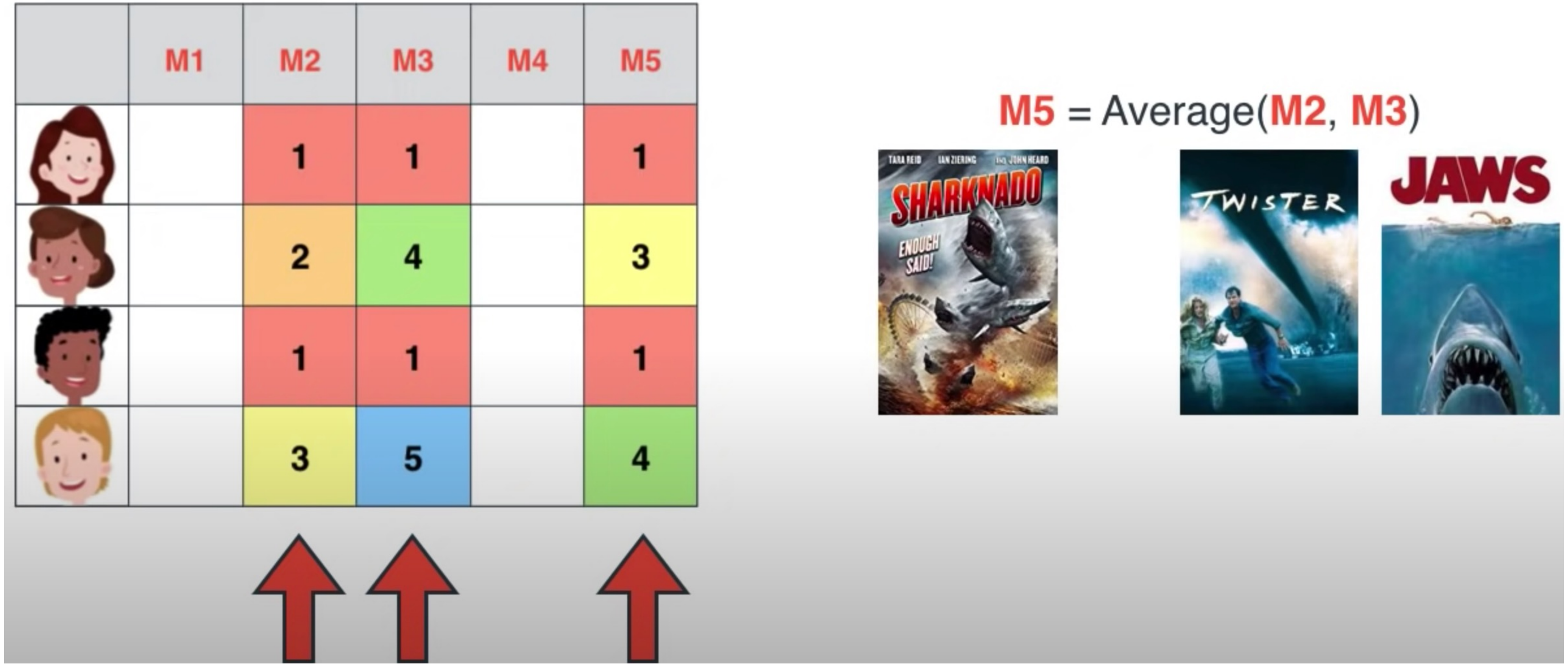
Полученные значение - латентные признаки - позволяют выстраивать персонализированные результаты, учитывать специфику фильмов и искать зависимости в данных, а также позволяют модели рассматривать непопулярные фильмы

Матричные разложения - интуиция



Полученные значение - латентные признаки - позволяют выстраивать персонализированные результаты, учитывать специфику фильмов и искать зависимости в данных, а также позволяют модели рассматривать непопулярные фильмы

Матричные разложения - интуиция



Полученные значение - латентные признаки - позволяют выстраивать персонализированные результаты, учитывать специфику фильмов и искать зависимости в данных, а также позволяют модели рассматривать непопулярные фильмы

Матричные разложения - интуиция

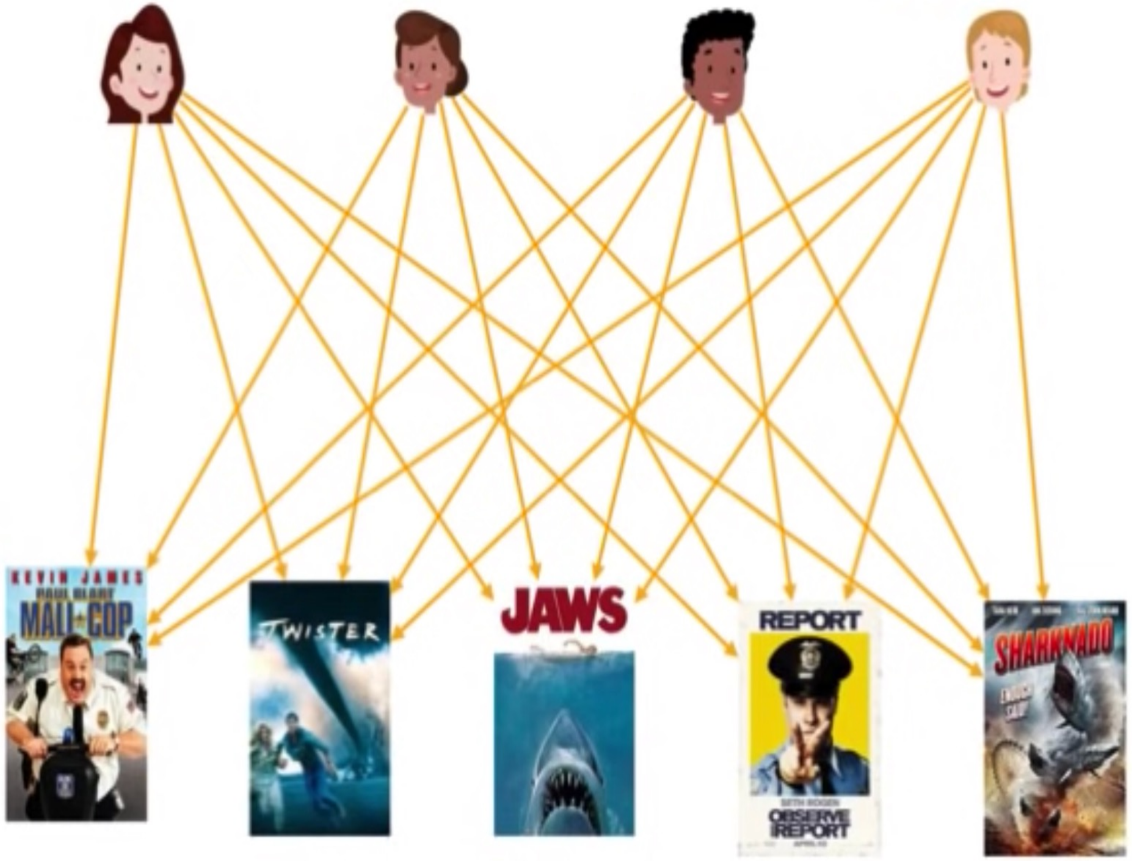
	M1	M2	M3	M4	M5
Comedy	3	1	1	3	1
Action	1	2	4	1	3

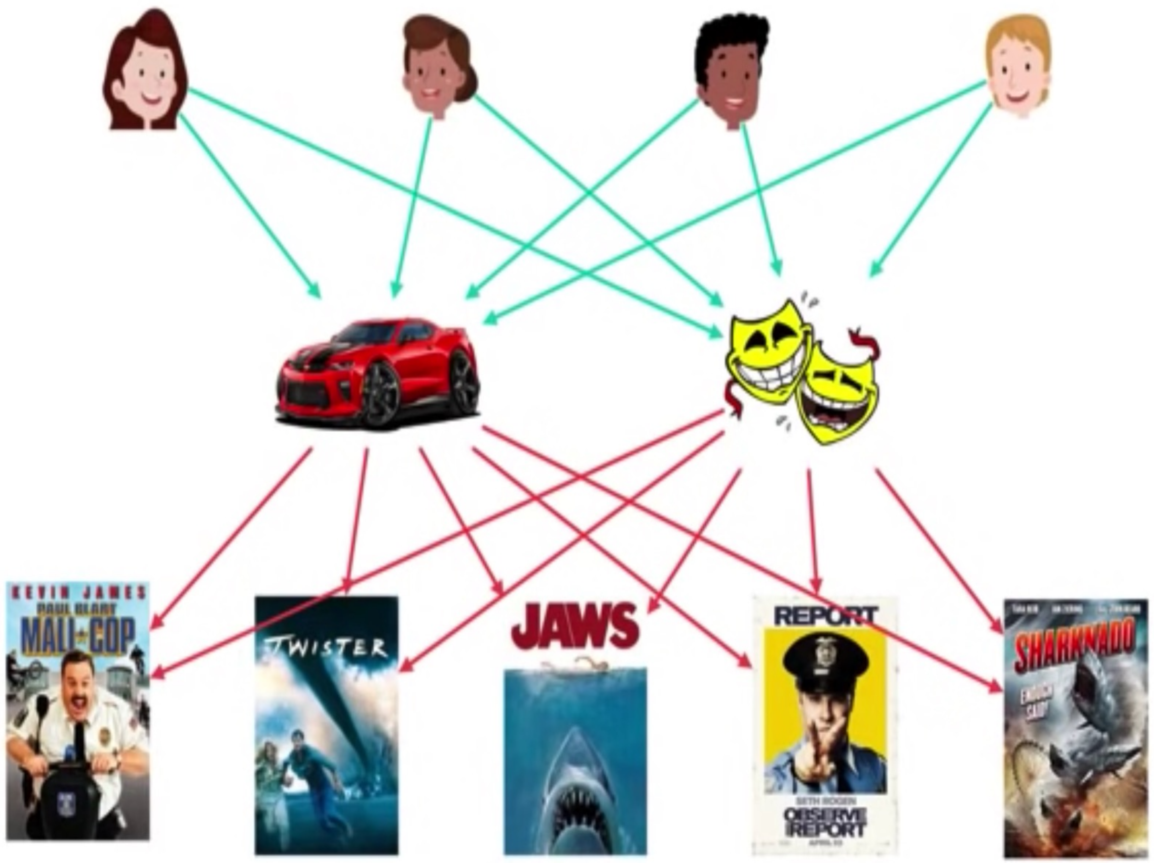
	Comedy	Action
A	✓	✗
B	✗	✓
C	✓	✗
D	✓	✓

	M1	M2	M3	M4	M5
A	3	1	1	3	1
B	1	2	4	1	3
C	3	1	1	3	1
D	4	3	5	4	4

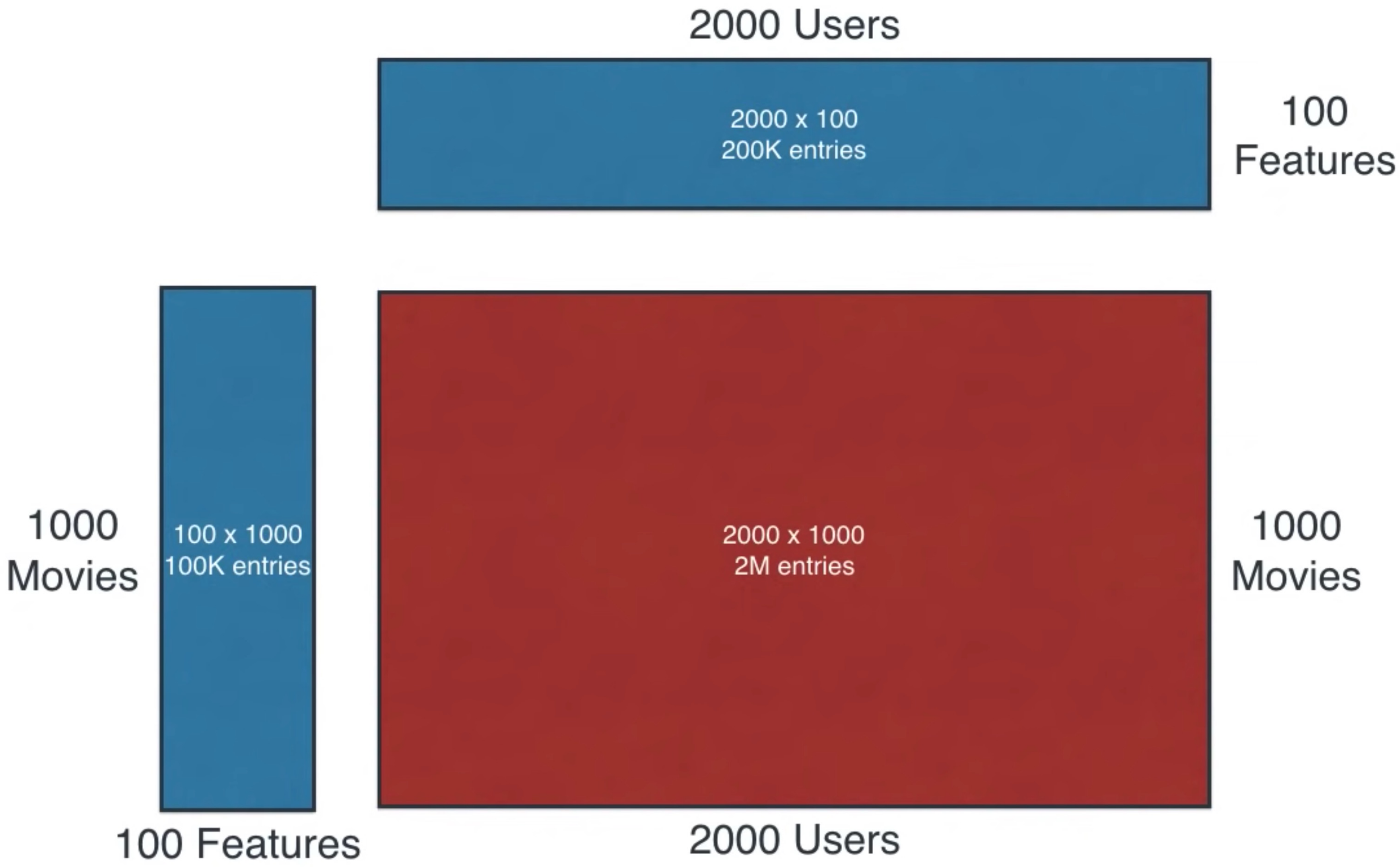
Полученные значение - латентные признаки - позволяют выстраивать персонализированные результаты, учитывать специфику фильмов и искать зависимости в данных, а также позволяют модели рассматривать непопулярные фильмы

Латентные признаки - интуиция





Матричные разложения - интуиция



Матричные разложения - как получать

- SVD - не масштабируется, не слишком гибкое, переобучится, пропуски
- rSVD - не масштабируется, параметризованное, переобучится, пропуски
- rSVD с регуляризацией - не масштабируется, параметризованное, может не переобучится, пропуски
- ALS с регуляризацией – масштабируется, параметризованное, может не переобучится, пропуски
- SGD с регуляризацией – масштабируется, параметризованное, может не переобучится, может игнорировать пропуски

А как обучаться то?

1. Делить пользователей на train/test – нехорошо, теряем пользователей
2. Делить фильмы на train/test – нехорошо, теряем фильмы
3. Выбрасывать одну часть точек в train, другую в test, обучаться восстанавливать самодельные пропуски

	M1	M2	M3	M4	M5
F1	3	1	1	3	1
F2	1	2	4	1	3

	F1	F2
A	1	0
B	0	1
C	1	0
D	1	1

	M1	M2	M3	M4	M5
A	3	1	1	3	1
B	1	2	4	1	3
C	3	1	1	3	1
D	4	3	5	4	4

А как предсказывать то?

1. $R \approx U @ M$ [Rating = User @ Movie] = R_{rec}, где R_{rec} – реконструированная (ожидаемая) матрица рейтингов, ограниченная рангом r.
2. Как вариант – брать для данного пользователя из соответствующей строки топ k еще не просмотренных фильмов

А почему это работает?

1. Давайте пробовать искать векторные представления u_i^T для пользователей и m_j для фильмов, такие что отмасштабированная оценка фильма j пользователем i это $\langle u_i^T, m_j \rangle$. То есть $R_{ij} = \langle u_i^T, m_j \rangle$
2. Тогда функционал потерь для нахождения имеет вид $L(P, Q) = ||R - UM||^2$ + регуляризация
3. При этом хорошо бы вычесть из R средние по столбцам и строкам, чтобы смотреть на «очищенные» данные
4. Теперь это просто задача приближения с регуляризацией и мы надеемся, что заполненные приближением P значения будут предсказаниями

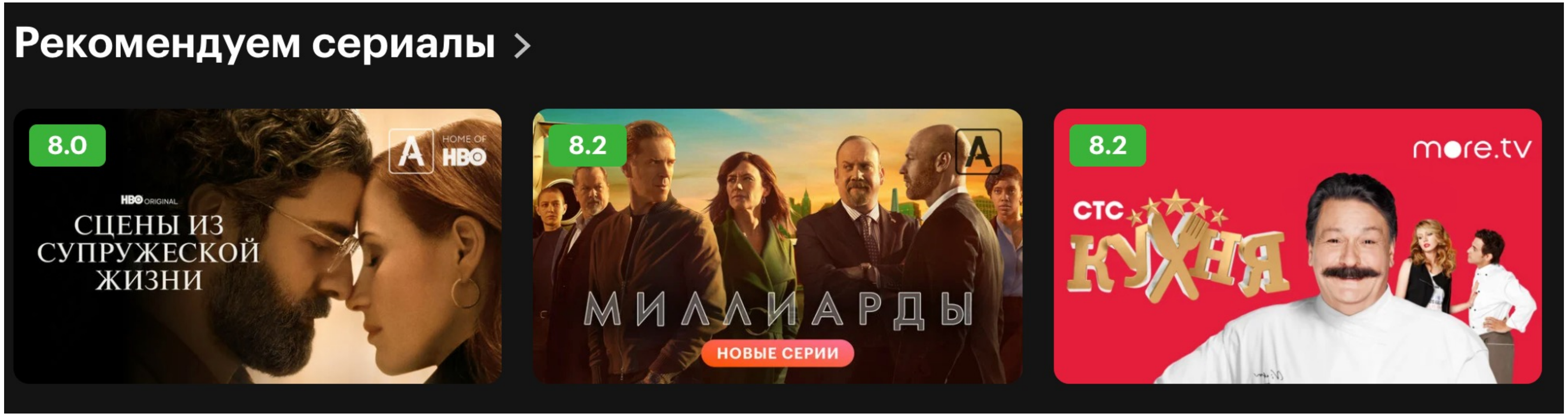
А что с проблемами?

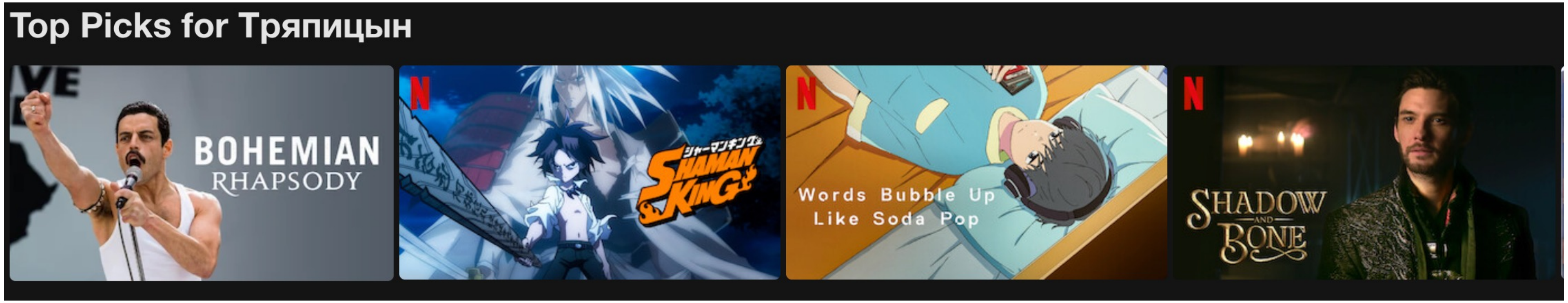
1. Вычислительная сложность:
   - рантайм - $O(|M|r) \ll O(|M||U|)$
   - предподсчет - $O(|M||U|r) \ll O(|M|^2|U|)$
2. Новые фильмы не будут иметь оценок, пока их не посмотрят много пользователей – ~~рекомендации по этому фильму будут плохими, а сам фильм почти не будем рекомендовать~~ Уже по малому числу оценок сможем выделять свойства фильма и рекомендовать его нужным пользователям
3. Популярные фильмы будем рекомендовать ~~чаще (по сути противоположность предыдущему пункту)~~ качественнее
4. Нет рекомендаций для новых пользователей
5. Выглядит ~~хиленько, никаких связей фильмов друг с другом и пользователей друг с другой не учитываем~~ уже гораздо интереснее

Какие интересные вопросы остались?

1. Есть ли разница какие точки выкидывать для обучения? Более старые для пользователя? Более новые?
2. Как мерить офлайн качество? Т.е. как с какой-то уверенностью, прежде чем катить модель в эксперимент, проверить (убедить менеджера), что она действительно хорошо работает? Что такое хорошо?
3. Как мерить дефект рейт в офлайне? Т.е. как с какой-то уверенностью, прежде чем катить модель в эксперимент, проверить (убедить менеджера), что у модели не слишком много стьюпидов? Как определить стьюпидность? Стьюпид в общем смысле не просто нерелевантный фильм, а фильм, при рекомендации которого в голове возникает «что за чушь мне рекомендуют». Это скорее про репутационные риски сервиса, чем про качество модели.
4. 2 и 3 вопросы относительно наивной реализации решается толокой? Но все еще вопрос - что такое хорошо, а что такое стьюпид?

~~Приглашаю работать в Кинопоиск~~





Список источников

- Материалы курса "Основы матричных вычислений", http://wiki.cs.hse.ru/Основы_матричных_вычислений_2020/2021

- Golub, G. ; Kahan, W.: Calculating the Singular Values and Pseudo-Inverse of a Matrix (1965), https://www3.math.tu-berlin.de/Vorlesungen/SS14/MatricesGraphsPDEs/paper_for_students/GolubKahanSVD.pdf

- Nicolas Gillis : The Why and How of Nonnegative Matrix Factorization, https://arxiv.org/pdf/1401.5226.pdf

Список источников

- https://builtin.com/data-science/step-step-explanation-principal-component-analysis
- https://medium.com/analytics-vidhya/topic-modeling-with-non-negative-matrix-factorization-nmf-3caf3a6bb6da
- https://pub.towardsai.net/machine-learning-dimensionality-reduction-via-principal-component-analysis-1bdc77462831
- https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05

Список источников

- https://www.youtube.com/watch?v=ZspR5PZemcs