

Вебинар

# Рекомендательные системы. Нейронные сети в задачах классификации.

Доррер Михаил Георгиевич,  
кандидат технических наук, доцент

# Рекомендательные системы

- Общая идея рекомендательных систем
- Content - based RS (рекомендации по содержанию)
- Similarity based RS (рекомендации по принципу сходства)
- Сингулярное разложение матрицы (SVD)

# Введение в рекомендательные системы

**Рекомендательные системы** — программы, которые пытаются предсказать, какие объекты будут интересны пользователю, имея определенную информацию о его профиле.

Прогноз реакции  
на элемент

Предложение других элементов,  
которые могут понравиться



Глобальные оценки; особенности и предпочтения, не меняющиеся месяцами или годами; интересные страницы; зависимость от характерных пользовательских черт: пол, место проживания и т.п.



Кратковременные тренды и быстрые изменения интересов во времени.

# Сбор данных

## Явные способы

- запрос у пользователя оценки объекта по дифференцированной шкале;
- запрос у пользователя ранжировки группы объектов от наилучшего к наихудшему;
- предъявление пользователю двух объектов с вопросом о том, какой из них лучше;
- предложение создать список объектов, любимых пользователем.

- Пользователи не всегда соглашаются на запросы
- Нужно придумывать уловки и/или предлагать что-то в обмен.

## Неявные способы

- наблюдение за тем, что осматривает пользователь в интернет-магазинах или базах данных другого типа;
- ведение записей о поведении пользователя онлайн;
- отслеживание содержимого компьютера пользователя.

- Неизвестно понравилось ли в конечном итоге человеку предложение
- Сложно предположить, что конкретно подтолкнуло человека к совершению покупки.

# Решаемые задачи

- Сокращение времени поиска нужных товаров или услуг;
- Повышение вероятности совершения сопутствующих целевых действий (купил телефон + чехол к нему);
- Минимизация времени поиска повышает лояльность потребителя к платформе;
- Можно использовать в приложениях, интернет-магазинах, онлайн-кинотеатрах и других сервисах, которые предлагают какой-то контент, товары или услуги;
- Удержание текущих потребителей, привлечении новых и повышении прибыли.

Подобрали для вас

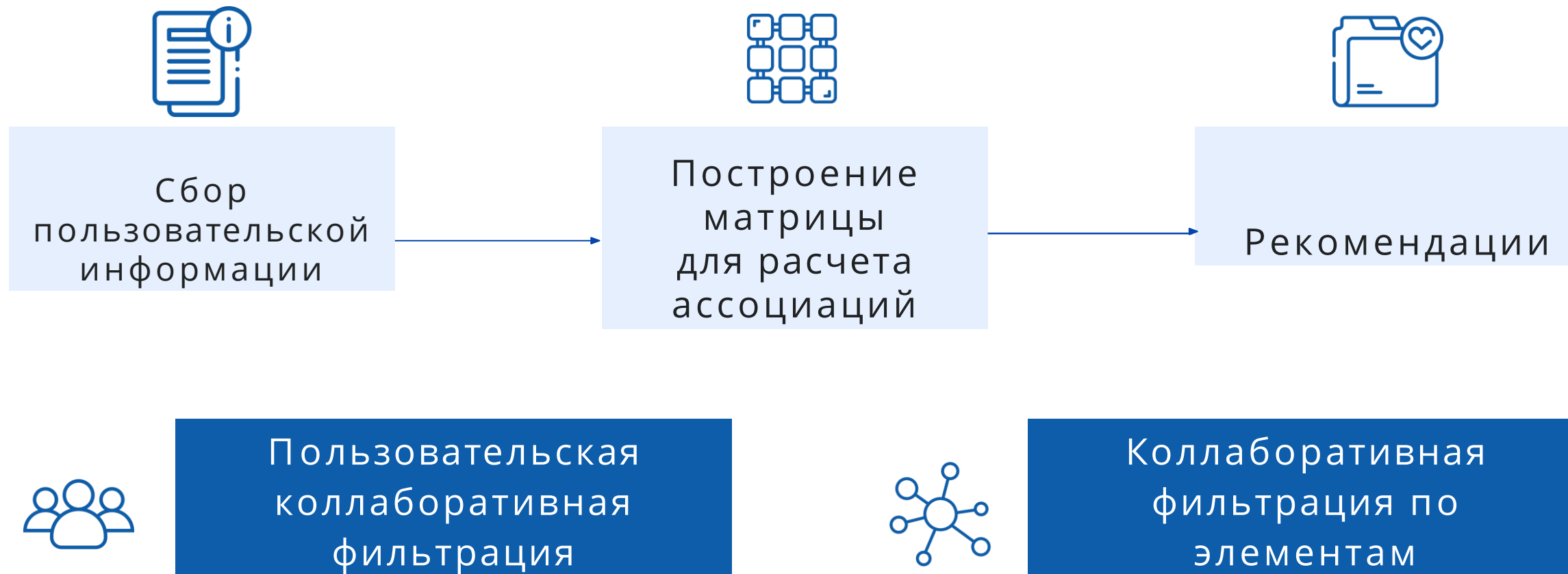
									
19 790 Р	44 990 Р 49 990 Р	74 620 Р 97 970 Р	38 990 Р	10 190 Р	19 790 Р	44 990 Р 49 990 Р	74 620 Р 97 970 Р	380 Р	
Беспроводные наушники Apple AirPo...	Швейная машина Janome Clio 325	Часы SUUNTO 9 Baro	Вышивальная машина Janome Memory Craft...	Распошивальная машина Janome Cover...	Беспроводные наушники Apple AirPo...	Швейная машина Janome Clio 325	Часы SUUNTO 9 Baro	Вышивальная машина Janome Memory Craft...	Смесь Nutr 1 Premium

# Основные проблемы

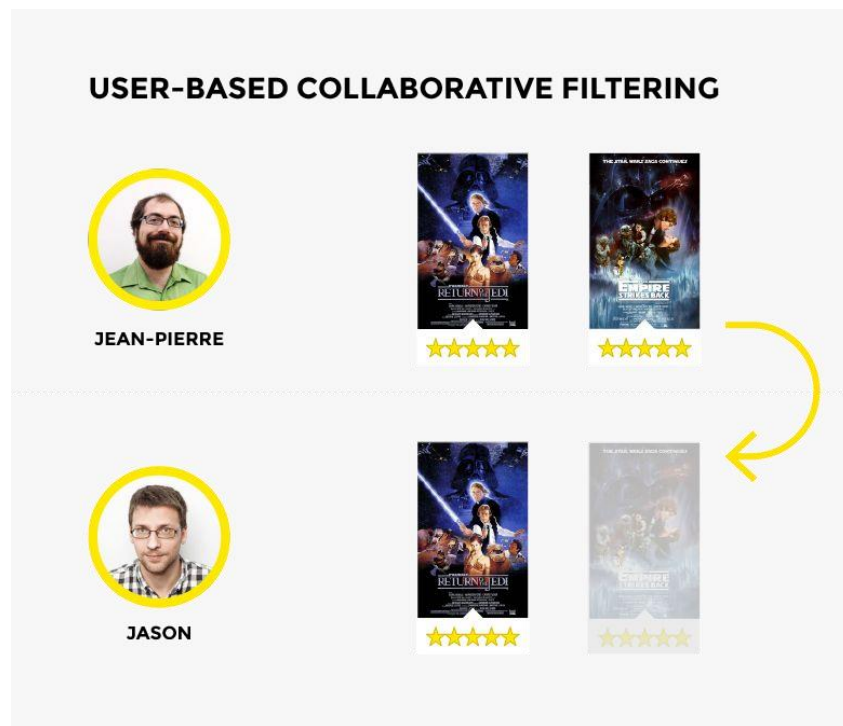
- Проблема холодного старта
  - На каждого нового пользователя или предмета невозможно сделать точные рекомендации. недостаточно данных и
- Масштабируемость
  - С увеличением количества пользователей в системе растет сложность расчетов.
- Разреженность данных
  - Большинство пользователей не ставит оценки товарам. Матрица «предмет- пользователь» получается очень большой и разреженной, что представляет проблемы при вычислении рекомендаций.

# Коллаборативная фильтрация

★ Простейшая и эффективная



# Пользовательская коллаборативная фильтрация



**USER / MOVIE MATRIX**

	GLADIATEUR	ROCKY IV	BEN-HUR	SKYFALL	IDIOCRACY
1	★☆☆☆☆			★★★★★	★★★☆☆
2		★★★★★		★★★★★	
3	★★★★★	★★★☆☆		★☆☆☆☆	
4			★★★☆☆	★★★★★	
5	★★★★★		★★★★★		

**Идея:** искать пользователей, чьи вкусы похожи на предпочтения нашего целевого пользователя.

1. Жан-Пьер и Джейсон поставили схожие оценки нескольким фильмам.
2. Их вкусы подобны.
3. Можем угадать неизвестные рейтинги Джейсона.

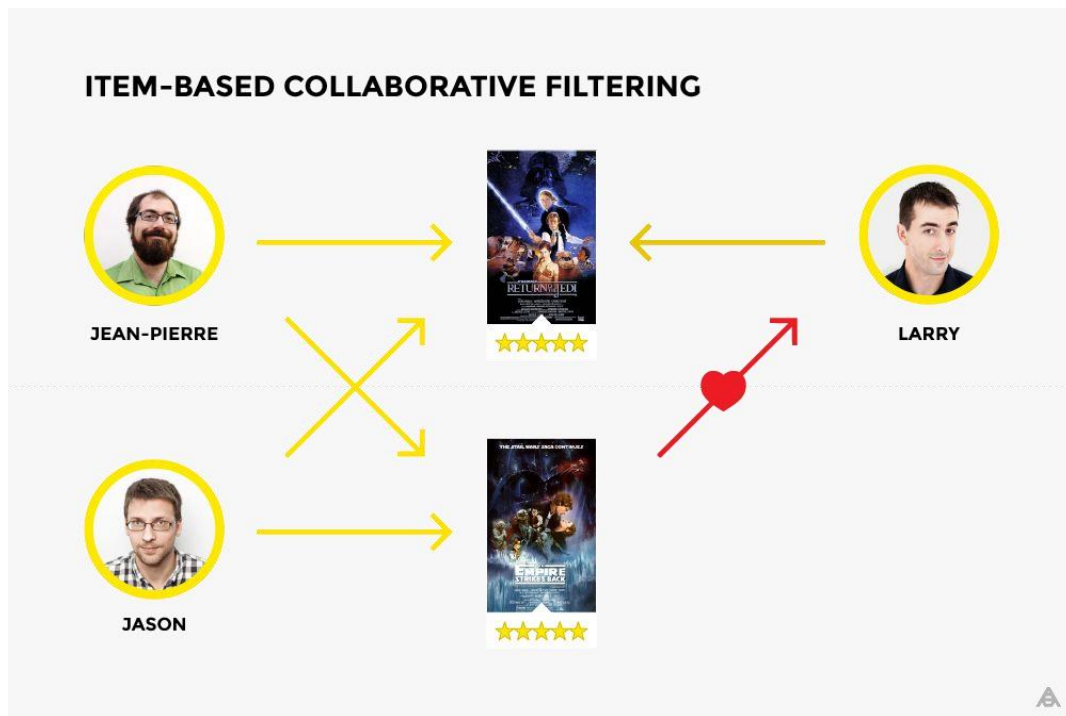


# Проблемы пользовательской коллаборативной фильтрации

- Пользовательские предпочтения со временем меняются. В результате система может генерировать множество неактуальных рекомендаций;
- Чем больше количество пользователей, тем больше времени уходит на генерирование рекомендации
- Фильтрация пользователей уязвима для накрутки рейтингов, когда злоумышленник обманывает систему и необъективно повышает оценку одних продуктов по сравнению с другими.

# Коллаборативная фильтрация по элементам

**Идея:** сходство двух элементов рассчитывается по рейтингам, выставленным пользователем.



1. Жан-Пьеру и Джейсону понравились оба фильма.
2. Можно сделать вывод, что большинству пользователей, высоко оценивших первый фильм, должен понравиться и второй.
3. Релевантно предложить фильм «Империя наносит ответный удар» Ларри, которому понравился фильм «Возвращение джедая».

# Достоинства коллаборативной фильтрации по элементам

- Лишена всех недостатков, присущих пользовательской фильтрации.
- Элементы в системе не изменяются со временем, поэтому рекомендации получатся более релевантными.
- Элементов обычно гораздо меньше, чем пользователей - обработка данных при такой фильтрации происходит быстрее.
- Такие системы гораздо сложнее обмануть.

# Входные данные - матрица оценок

		идентификатор продукта								ИМЯ ПОЛЬЗОВАТЕЛЯ
		1	2	3	4	5	6	7	8	9
alex		5.0000	3.0000			4.0000				
ivan		4.0000					1.0000		2.0000	3.0000
bob			5.0000	5.0000						
david				4.0000	3.0000		2.0000	1.0000		
		выставленная пользователем оценка								

Разработаем функцию, которая  
прочитает данный csv-файл.

alex,1,5.0  
alex,2,3.0  
alex,5,4.0  
ivan,1,4.0  
ivan,6,1.0  
ivan,8,2.0  
ivan,9,3.0  
bob,2,5.0  
bob,3,5.0  
david,3,4.0  
david,4,3.0  
david,6,2.0  
david,7,1.0

```
import csv

def ReadFile (filename = "<csv_file_location>"):

    f = open (filename)

    r = csv.reader (f)

    mentions = dict()

    for line in r:

        user      = line[0]

        product   = line[1]

        rate      = float(line[2])

        if not user in mentions:

            mentions[user] = dict()

        mentions[user][product] = rate

    f.close()

    return mentions
```

Для хранения рекомендаций  
будем использовать  
стандартную для python  
структуру данных dict: каждому  
пользователю ставится в  
соответствие справочник его  
оценок вида «продукт»:«оценка».

# Мера подобия

- Для рекомендации пользователю №1 какого-либо продукта, выбирать нужно из продуктов, которые нравятся каким-то пользователям 2-3-4-etc., которые наиболее похожие по своим оценкам на пользователя №1.
- Как получить численное выражение этого «подобия» («сходства», «похожести») пользователей?
- Допустим, у нас есть  $M$  продуктов. Оценки, выставленные отдельно взятым пользователем, представляют собой вектор в  $M$ -мерном пространстве продуктов.

★ Используются  
чаще всего

## Меры подобия:

1. Косинусная мера
2. Коэффициент корреляции Пирсона
3. Евклидово расстояние
4. Коэффициент Танимото
5. Манхэттенское расстояние и т.д.

# Косинусная мера

$$\cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{||\vec{x}||_2 \times ||\vec{y}||_2}$$

Скалярное произведение вектора самого на себя дает квадрат длины вектора.

```
def distCosine (vecA, vecB):  
    def dotProduct (vecA, vecB):  
        d = 0.0  
        for dim in vecA:  
            if dim in vecB:  
                d += vecA[dim]*vecB[dim]  
        return d  
    return dotProduct (vecA,vecB) /  
    math.sqrt(dotProduct (vecA,vecA)) /  
    math.sqrt(dotProduct (vecB,vecB))
```

# Алгоритм коллаборативной фильтрации

	<b>alex</b>	<b>bob</b>	<b>david</b>	<b>sum</b>
<b>ivan</b>	0.5164	0.0000	0.0667	0.5831

Для каждого из пользователей  
умножить его оценки на  
вычисленную величину меры, таким  
образом оценки более «похожих»  
пользователей будут сильнее  
влиять на итоговую позицию  
продукта

<p>Для каждого из продуктов посчитать сумму калиброванных оценок L наиболее близких пользователей, полученную сумму разделить на сумму мер L выбранных пользователей.</p>		<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>
	<b>alex</b>	2.5820	1.5492	0.0000	0.0000	2.0656	0.0000	0.0000	0.0000	0.0000
	<b>bob</b>	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	<b>david</b>	0.0000	0.0000	0.2668	0.2001	0.0000	0.1334	0.0667	0.0000	0.0000
	<b>sum</b>	2.5820	1.5492	0.2668	0.2001	2.0656	0.1334	0.0667	0.0000	0.0000
	<b>result</b>	<b>4.4281</b>	<b>2.6568</b>	<b>0.4576</b>	<b>0.3432</b>	<b>3.5424</b>	<b>0.2288</b>	<b>0.1144</b>	<b>0.0000</b>	<b>0.0000</b>



# Алгоритм коллаборативной фильтрации в виде формулы

$$r_{u,i} = k \sum_{u' \in U} sim(u, u') r_{u',i}$$

где функция **sim** — выбранная нами мера схожести двух пользователей,  
**U** — множество пользователей, **r** — выставленная оценка, **k** —  
нормировочный коэффициент:

$$k = 1 / \sum_{u' \in U} |sim(u, u')|$$

```

import math
def makeRecommendation (userID, userRates, nBestUsers,
nBestProducts):
    matches = [(u, distCosine(userRates[userID],
userRates[u])) for u in userRates if u != userID]
    bestMatches = sorted(matches, key= lambda (x,y):(y,x),
reverse=True)[:nBestUsers]
    print "Most correlated with '%s' users:" % userID
    for line in bestMatches:
        print "  UserID: %6s  Coeff: %6.4f" % (line[0],
line[1])
    sim = dict()
    sim_all = sum([x[1] for x in bestMatches])
    bestMatches = dict([x for x in bestMatches if x[1] >
0.0])
    for relatedUser in bestMatches:
        for product in userRates[relatedUser]: if
not product in userRates[userID]:
            if not product in sim:
                sim[product] = 0.0
            sim[product] +=
userRates[relatedUser][product] * bestMatches[relatedUser]
        for product in sim:
            sim[product] /= sim_all
    bestProducts = sorted(sim.iteritems(), key=lambda
(x,y):(y,x), reverse= True)[:nBestProducts]
    print "Most correlated products:"
    for prodInfo in bestProducts:
        print "  ProductID: %6s  CorrelationCoeff: %6.4f" %
(prodInfo[0], prodInfo[1])
    return [(x[0],x[1]) for x in bestProducts]

```

Для проверки его работоспособности  
можно выполнить следующую  
команду:

```

rec = makeRecommendation ('ivan',
ReadFile(), 5, 5)

```

Что приведет к следующему  
результату:

```

Most correlated with 'ivan' users:
  UserID:  alex  Coeff: 0.5164
  UserID:  david Coeff: 0.0667
  UserID:   bob  Coeff: 0.0000
Most correlated products:
  ProductID: 5  CorrelationCoeff: 3.5426
  ProductID: 2  CorrelationCoeff: 2.6570
  ProductID: 3  CorrelationCoeff: 0.4574
  ProductID: 4  CorrelationCoeff: 0.3430
  ProductID: 7  CorrelationCoeff: 0.1143

```

# Возможные пути оптимизации и доработки

## Оптимизация используемых структур данных

При хранении данных в python в виде dict, при каждом обращении к конкретному значению производится вычисление хэша и ситуация становится тем хуже, чем длиннее строки названий.

- В практических задачах для хранения данных можно использовать разреженные матрицы
- Вместо текстовых имен пользователей и названий продуктов использовать числовые идентификаторы - пронумеровать всех пользователей и все продукты

# Возможные пути оптимизации и доработки

- Оптимизация производительности
  - Вычислять рекомендацию при каждом обращении пользователя — занятие крайне затратное.
- 1. Кластеризация пользователей и вычисление меры схожести только между пользователями, принадлежащими одному кластеру;
- 2. Вычисление коэффициентов схожести продукт-продукт:
  - Транспонировать матрицу пользователь-продукт в матрицу продукт-пользователь;
  - Для каждого из продуктов вычислить набор наиболее схожих продуктов, используя косинусную меру и запомнив  $k$  ближайших. Этот процесс можно производить 1 раз в  $M$  часов/дней.
  - Умножить оценки пользователя на значение меры схожести продуктов.
  - Получим рекомендацию за  $O(N*k)$ , где  $N$  — количество оценок пользователя.

# Возможные пути оптимизации и доработки

## Оптимизация +

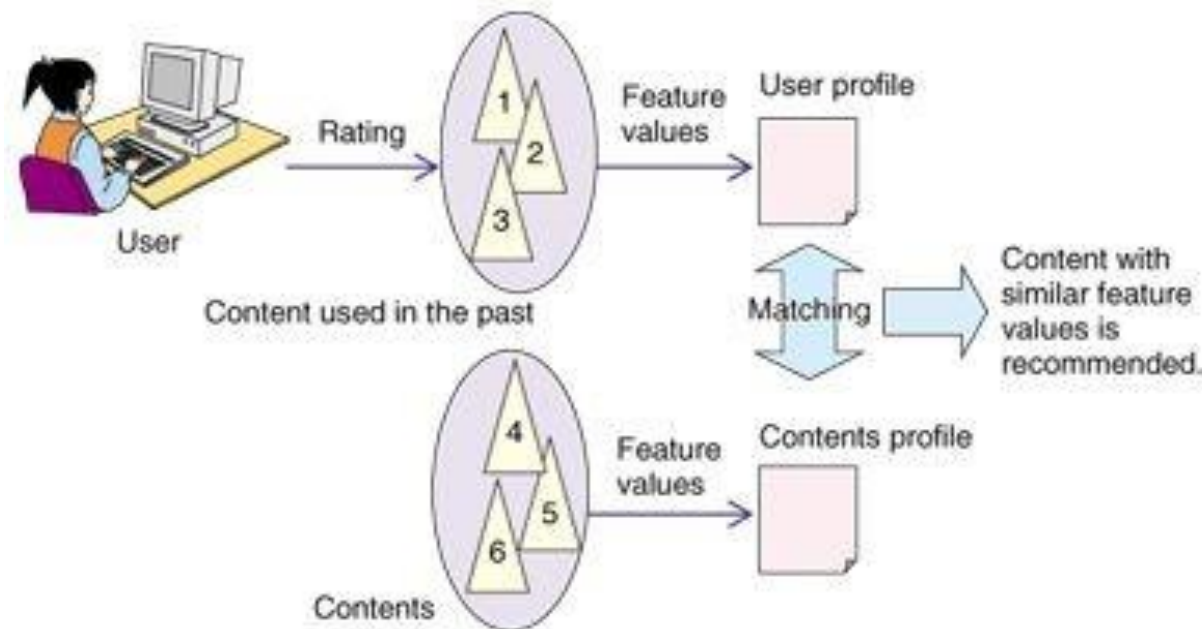
- **Подбор иной меры сходства**
  - ! Выбор меры нужно производить только по результатам анализа данных системы
- **Модификация алгоритма фильтрации**
- Другой алгоритм фильтрации может дать более точные рекомендации в конкретной системе
  - ! Сравнение различных алгоритмов можно производить только в применении к конкретной системе

# Коллаборативная фильтрация. Основные обозначения

- Начнём краткий обзор разных рекомендательных систем с коллаборативной фильтрации.
- Обозначения:
  - индекс  $i$  всегда будет обозначать пользователей (всего пользователей будет  $N, i = 1..N$ );
  - индекс  $a$  – предметы (сайты, товары, фильмы...), которые мы рекомендуем (всего  $M, a = 1..M$ );
  - $x_i$  – набор (вектор) признаков (features) пользователя,  $x_a$  – набор признаков предмета;
  - когда пользователь  $i$  оценивает предмет  $a$ , он производит отклик (response, rating)  $r_{i,a}$ ; этот отклик – случайная величина, конечно.
- Наша задача – предсказывать оценки  $r_{i,a}$ , зная признаки  $x_i$  и  $x_a$  для всех элементов базы и зная некоторые уже расставленные в базе  $r_{i',a'}$ .
- Предсказание будем обозначать через  $\hat{r}_{i,a}$ .

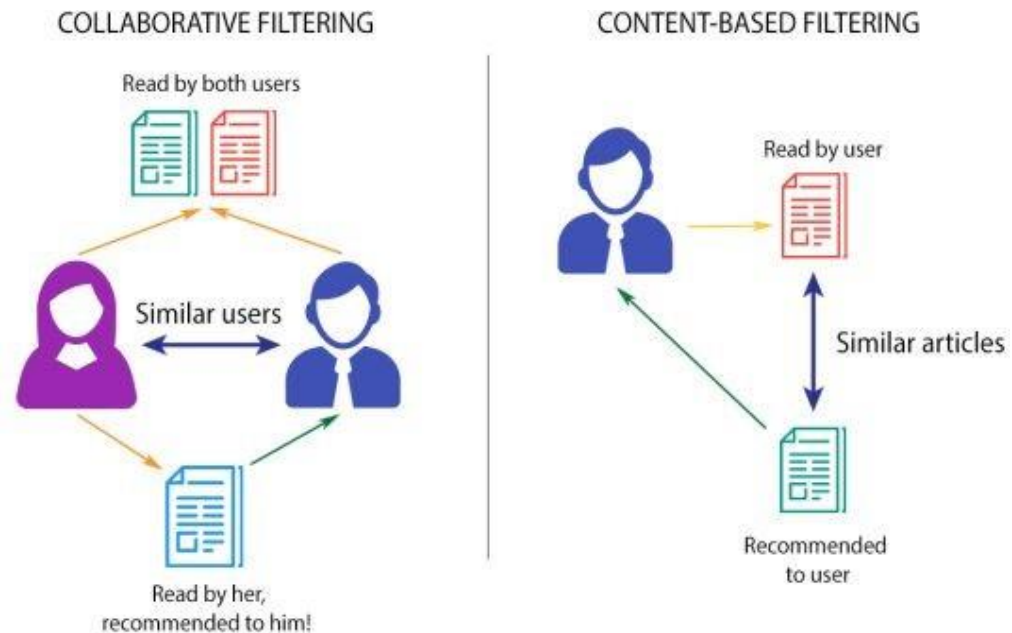
# Идея content - based рекомендательной системы

1. Анализ контента
2. Изучение профиля пользователя
3. Фильтрация компонентов связывает характерные черты профиля пользователя с чертами продукта



# Применение content-based рекомендательных систем

- Не использует информацию о предпочтениях других пользователей, принимая во внимания лишь его атрибуты и историю взаимодействия с сущностями
- Работает при недостатке информации о взаимодействии других пользователей с сущностями
- Работает при наличии обширных данных о самих пользователях





# Достоинства и недостатки content based RS

## Достоинства

1. Независимость пользователя;
2. Точность метода;
3. Способность давать рекомендации; пользователям с уникальным вкусом;
4. Способность рекомендовать новые и непопулярные продукты;
5. Прозрачность.

## Недостатки

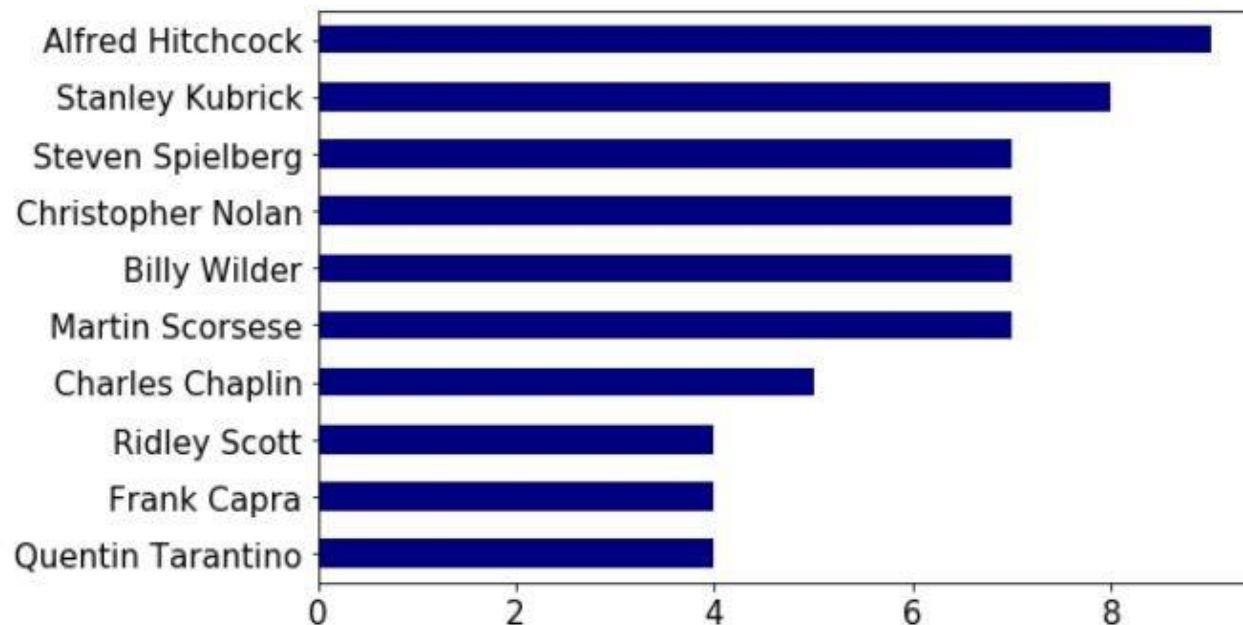
1. Ограниченный контент анализ;
2. Невозможно предсказать, что точно хочет пользователь, если он не выказал в этом интерес;
3. Сверхспециализация;
4. Невозможно использовать качественную оценку других пользователей о продукте;
5. Недостаток данных.

# Загрузка данных и подготовка

```
from rake_nltk import Rake
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import
cosine_similarity
from sklearn.feature_extraction.text
import CountVectorizer
df =
pd.read_csv('IMDB_Top250Engmovies2_OMDB
_Detailed.csv')
df.head()
```

```
df['Director'].value_counts()[0:10].plo
t('barh', figsize=[8,5], fontsize=15,
color='navy').invert_yaxis()
```

- Название
- Режиссер
- Актеры
- Сюжет
- Жанр



# Подготовка данных

- удаление стоп-слов
- удаление пунктуации
- удаление пробелов
- конвертации всех слов в нижний регистр

```
df['Key_words'] = ''
r = Rake()
for index, row in df.iterrows():
    r.extract_keywords_from_text(row['Plot'])
    key_words_dict_scores = r.get_word_degrees()
    row['Key_words'] =
list(key_words_dict_scores.keys())
```

## Использование функции Rake для извлечения ключевых слов

Plot

Two imprisoned men bond over a number of years, finding ...

The aging patriarch of an organized crime dynasty transf...

The early life and career of Vito Corleone in 1920s New ...

When the menace known as the Joker emerges from his myst...

A jury holdout attempts to prevent a miscarriage of just...



Key\_words

[finding, solace, years, acts, number, eventual, redempt...

[aging, patriarch, organized, crime, dynasty, transfers,...

[portrayed, 1920s, new, york, family, crime, syndicate, ...

[mysterious, past, gotham, ability, dark, knight, must, ...

[miscarriage, jury, holdout, attempts, colleagues, justi...



# Подготовка данных

```
df['Genre'] = df['Genre'].map(lambda x: x.split(','))
df['Actors'] = df['Actors'].map(lambda x: x.split(',')[:3])
df['Director'] = df['Director'].map(lambda x: x.split(','))
for index, row in df.iterrows():
    row['Genre'] = [x.lower().replace(' ', '') for x in row['Genre']]
    row['Actors'] = [x.lower().replace(' ', '') for x in row['Actors']]
    row['Director'] = [x.lower().replace(' ', '') for x in row['Director']]
```

Director	Actors	Genre
Frank Darabont	Tim Robbins, Morgan Freeman, Bob Gunton, William Sadler	Crime, Drama
Francis Ford Coppola	Marlon Brando, Al Pacino, James Caan, Richard S. Castellano	Crime, Drama
Francis Ford Coppola	Al Pacino, Robert Duvall, Diane Keaton, Robert De Niro	Crime, Drama
Christopher Nolan	Christian Bale, Heath Ledger, Aaron Eckhart, Michael Caine	Action, Crime, Drama
Sidney Lumet	Martin Balsam, John Fiedler, Lee J. Cobb, E.G. Marshall	Crime, Drama



Director	Actors	Genre
[frankdarabont]	[timrobbins, morganfreeman, bobgunton]	[crime, drama]
[francisfordcoppola]	[marlonbrando, alpacino, jamescaan]	[crime, drama]
[francisfordcoppola]	[alpacino, robertduvall, dianekeaton]	[crime, drama]
[christophernolan]	[christianbale, heathledger, aaroneckhart]	[action, crime, drama]
[sidneylumet]	[martinbalsam, johnfiedler, lee.j.cobb]	[crime, drama]

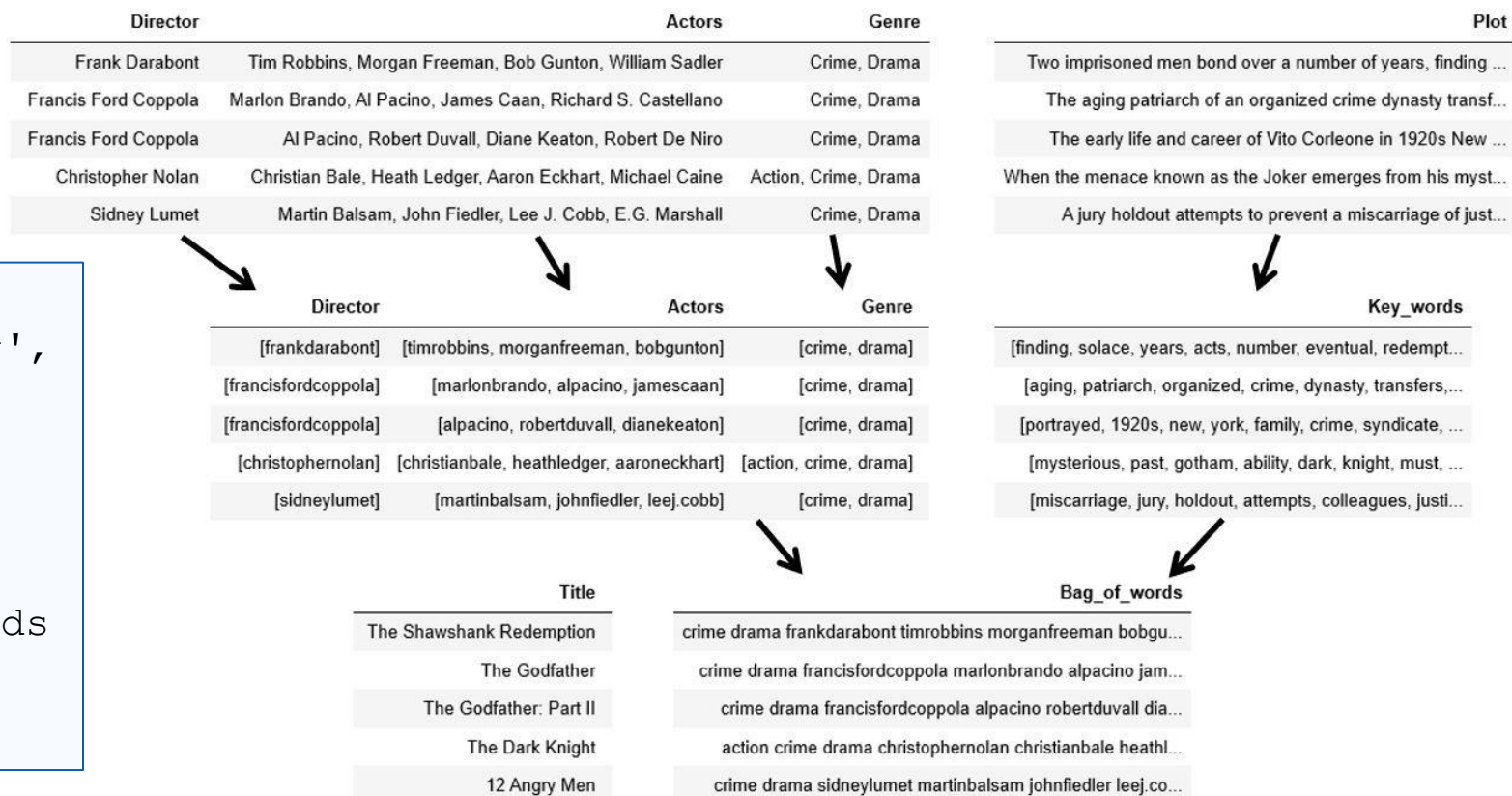
Все имена трансформированы в уникальные идентификационные значения



# Подготовка данных

```
df['Bag_of_words'] = ''
columns = ['Genre', 'Director',
           'Actors', 'Key_words']
for index, row in df.iterrows():
    words = ''
    for col in columns:
        words += ' ' + row[col]
    row['Bag_of_words'] = words

df = df[['Title', 'Bag_of_words']]
```



# Подготовка данных

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|} = \frac{\sum_{i=1}^n u_i v_i}{\sqrt{\sum_{i=1}^n u_i^2} \sqrt{\sum_{i=1}^n v_i^2}}$$

$$\mathbf{u} \cdot \mathbf{v} = [u_1 \ u_2 \ \dots \ u_n] \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = u_1 v_1 + u_2 v_2 + \dots + u_n v_n = \sum_{i=1}^n u_i v_i$$

Формула косинусного коэффициента для расчета значений в подобных матрицах

```
count = CountVectorizer()
count_matrix = count.fit_transform(df['Bag_of_words'])
cosine_sim = cosine_similarity(count_matrix, count_matrix)
print(cosine_sim)
```

30



# Подготовка данных

	Movie0	Movie1	Movie2	...	Movie247	Movie248	Movie249
Movie0	[1.	0.15789474	0.13764944	...	0.05263158	0.05263158	0.05564149]
Movie1	[0.15789474	1.	0.36706517	...	0.05263158	0.05263158	0.05564149]
Movie2	[0.13764944	0.36706517	1.	...	0.04588315	0.04588315	0.04850713]
...	...						
Movie247	[0.05263158	0.05263158	0.04588315	...	1.	0.05263158	0.05564149]
Movie248	[0.05263158	0.05263158	0.04588315	...	0.05263158	1.	0.05564149]
Movie249	[0.05564149	0.05564149	0.04850713	...	0.05564149	0.05564149	1.]

Матрица подоби  
250 строк x 250 столбцов

Создать серию названий фильмов для того, чтобы список из названий совпадал со списком строк и столбцов матрицы подоби.

`indices = pd.Series(df['Title'])`

31

# Обучение алгоритма

```
def recommend(title, cosine_sim = cosine_sim):  
    recommended_movies = []  
    idx = indices[indices == title].index[0]  
    score_series = pd.Series(cosine_sim[idx]).sort_values(ascending = False)  
    top_10_indices = list(score_series.iloc[1:11].index)  
    for i in top_10_indices:  
        recommended_movies.append(list(df['Title'])[i])  
    return recommended_movies
```

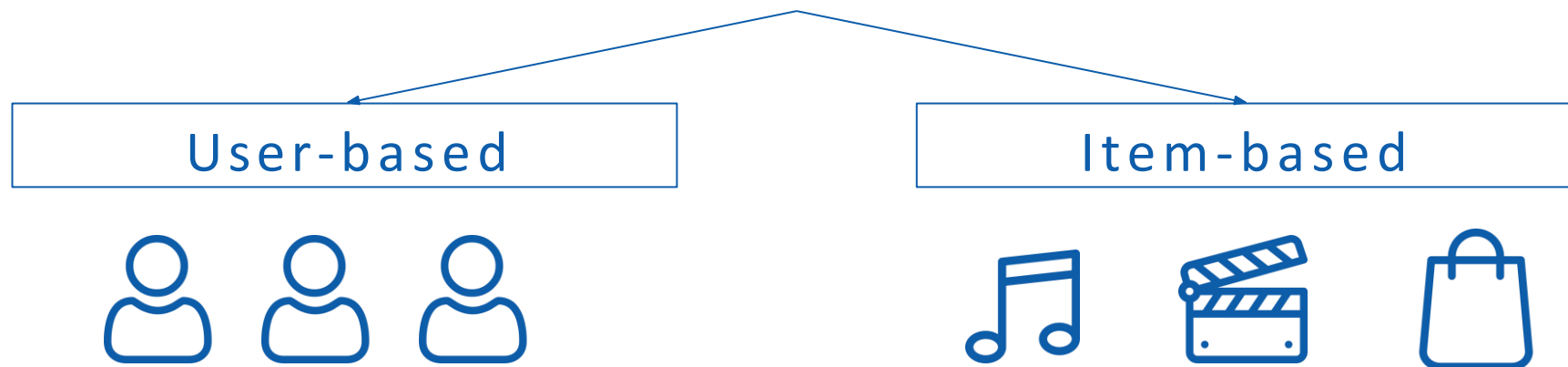
`recommend('The Avengers')`

`['Guardians of the Galaxy Vol. 2',  
'Aliens',  
'Guardians of the Galaxy',  
'The Martian',  
'Interstellar',  
'Blade Runner',  
'Terminator 2: Judgment Day',  
'The Thing',  
'The Terminator',  
'Spider-Man: Homecoming']`



# Similarity based RS

(рекомендации по принципу сходства)

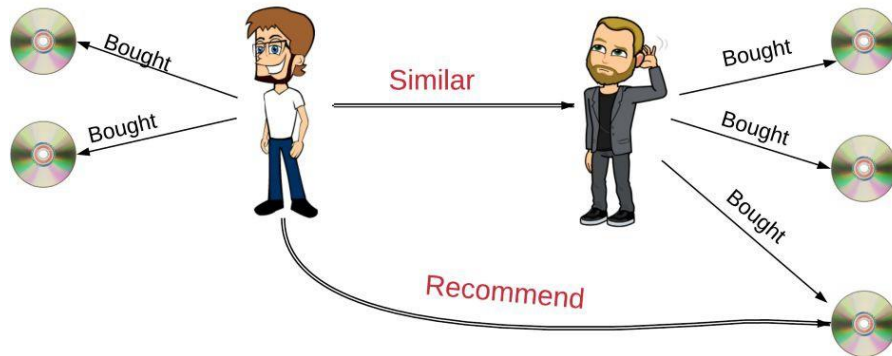


- Выберем некоторую условную меру схожести пользователей по их истории оценок  $sim(u, v)$ .
- Объединим пользователей в группы (кластеры) так, чтобы похожие пользователи оказывались в одном кластере:  $u \mapsto F(u)$ .
- Оценку пользователя объекту будем предсказывать как среднюю оценку кластера этому объекту:

$$\hat{r}_{ui} = \frac{1}{|F(u)|} \sum_{v \in F(u)} r_{vi}$$

# User-based

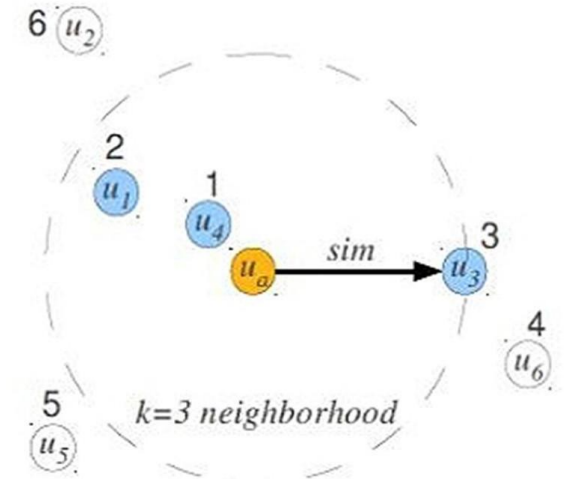
- Для каждого пользователя ищем
- $k$  наиболее похожих на него и дополняем информацию о пользователе известными данными по его соседям



	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$	$i_7$	$i_8$
$u_1$	?	4.0	4.0	2.0	1.0	2.0	?	?
$u_2$	3.0	?	?	?	5.0	1.0	?	?
$u_3$	3.0	?	?	3.0	2.0	2.0	?	3.0
$u_4$	4.0	?	?	2.0	1.0	1.0	2.0	4.0
$u_5$	1.0	1.0	?	?	?	?	?	1.0
$u_6$	?	1.0	?	?	1.0	1.0	?	1.0
$u_a$	?	?	4.0	3.0	?	1.0	?	5.0
$r_a$	3.5	4.0			1.3	2.0		

$$\hat{r}_{aj} = \frac{1}{\sum_{i \in N(a)} s_{ai}} \sum_{i \in N(a)} s_{ai} r_{ij}$$

Пользователь



$$\text{sim}_{\text{Cosine}}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$$

Его ближайшие соседи

# User-based

- Минус - плохо применим на практике из-за квадратичной сложности

$$O(n^2 m)$$

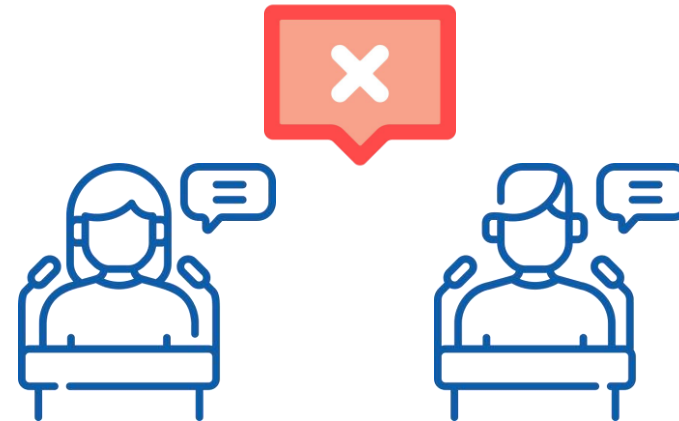
- где  $n$  — число пользователей, а  $m$  — число товаров.
- Возможные корректировки:
  - обновлять расстояния не при каждой покупке, а батчами (например, раз в день)
  - не пересчитывать матрицу расстояний полностью, а обновлять ее инкрементно
  - сделать выбор в пользу итеративных и приближенных алгоритмов (например, ALS).

# User-based

★ MovieLens

Оптимальное количество  
соседей 30-50 для фильмов и  
25-100 для произвольных  
рекомендаций

- Предположения:
  - Вкусы людей не меняются временем (или меняются, но для всех одинаково).
  - Если вкусы людей совпадают, то они совпадают во всем.



# Item-based

- Применяется к транспонированной матрице предпочтений, т.е. ищет близкие товары, а не пользователей.
- Предположения:
  - Ближайшие соседи ищутся на множестве товаров — столбцов матрицы предпочтений. И усреднение происходит именно по ним.
  - Если продукты содержательно похожи, то они либо одновременно нравятся, либо одновременно не нравятся.



# Преимущества item-based

- Когда пользователей много, задача поиска ближайшего соседа становится плохо вычислимой.
- Для 1 млн пользователей нужно рассчитать и хранить  $\frac{1}{2} 10^6 * 10^6 \sim 500$  млрд расстояний. Подход Item-based снижает сложность вычислений с  $O(N^2n)$  до  $O(n^2N)$
- Сильно снижается размерность матрицы расстояний: не (1 млн на 1 млн), а (100 на 100) при количестве товаров равном 100.
- Оценка близости товаров гораздо более точная, чем оценка близости пользователей. Пользователей обычно намного больше, чем товаров и следовательно стандартная ошибка при расчете корреляции товаров там существенно меньше.
- Описания пользователей сильно разрежены (товаров много, оценок мало) Оптимизация расчета — перемножаем только те элементы, где есть пересечение.
- Предпочтения пользователя могут меняться со временем, но описание товаров практически постоянно.

# Пути усовершенствования Item-based

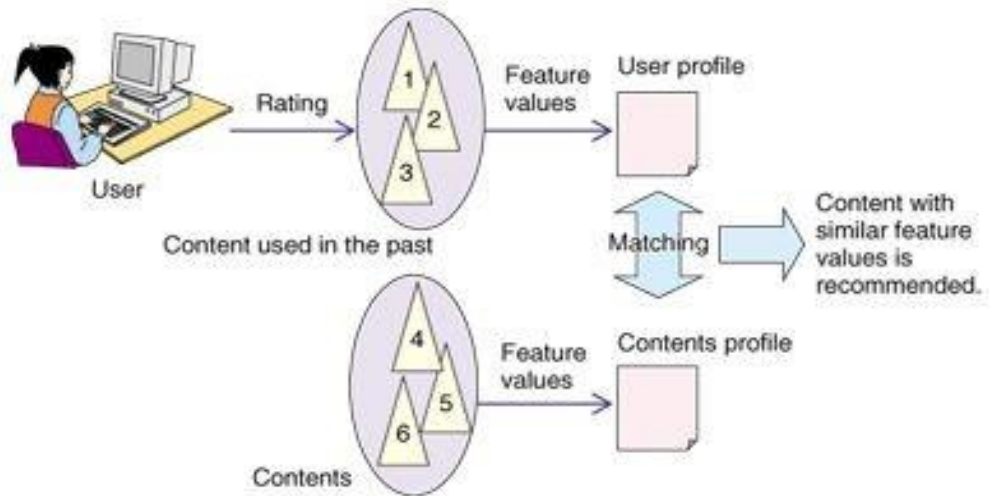
- Считать «похожесть» продуктов не типичным косинусным расстоянием, а сравнивая их содержание (content-based similarity). При этом вторая часть алгоритма — получение усредненных оценок — никак не изменяется.
- При вычислении item similarity взвешивать пользователей.
- Чем больше пользователь сделал оценок, тем больший у него вес при сравнении двух товаров.
- Вместо простого усреднения оценок по соседним товарам можно подбирать веса, делая линейную регрессию.

# Проблемы similarity based RS

- Нечего рекомендовать новым/нетипичным пользователям. Для таких пользователей не найдется подходящего кластера с похожими на них пользователями.
- Холодный старт — новые объекты никому не рекомендуются.
- Рекомендации часто тривиальны.
- Не учитывается специфика каждого пользователя. В каком-то смысле мы делим всех пользователей на какие-то классы (шаблоны).
- Если в кластере никто не оценивал объект, то предсказание сделать не получится.



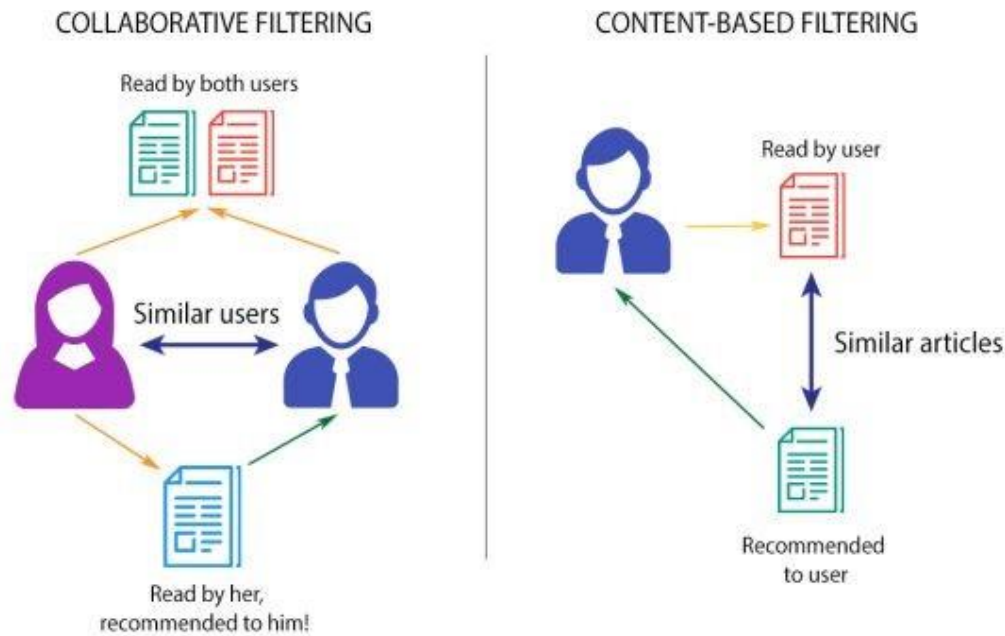
# Идея content - based рекомендательной системы



1. Анализ контента
2. Изучение профиля пользователя
3. Фильтрация компонентов связывает характерные черты профиля пользователя с чертами продукта

# Применение content-based рекомендательных систем

- Не использует информацию о предпочтениях других пользователей, принимая во внимания лишь его атрибуты и историю взаимодействия с сущностями
- Работает при недостатке информации о взаимодействии других пользователей с сущностями
- Работает при наличии обширных данных о самих пользователях



# Достоинства

1. Независимость от пользователя;
2. Точность метода;
3. Способность давать рекомендации; пользователям с уникальным вкусом;
4. Способность рекомендовать новые и непопулярные продукты;
5. Прозрачность.

# Недостатки

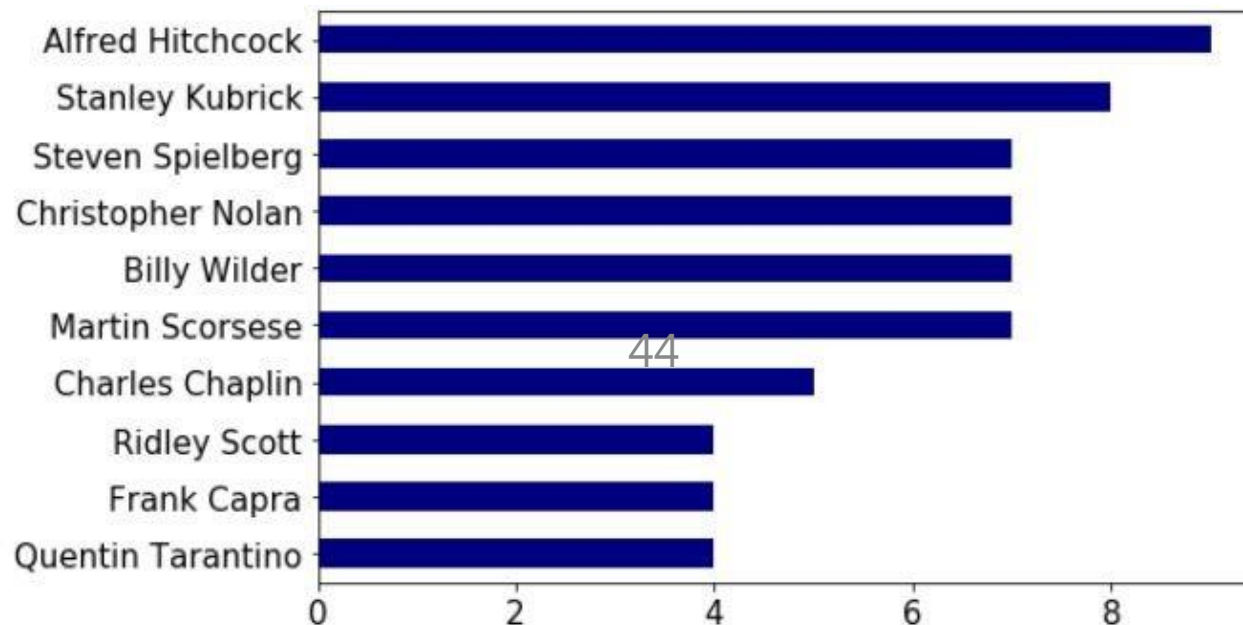
1. Ограниченный контент анализ;
2. Невозможно предсказать, что точно хочет пользователь, если он не выказал в этом интерес;
3. Сверхспециализация;
4. Невозможно использовать качественную оценку других пользователей о продукте;
5. Недостаток данных.

# Загрузка данных и подготовка

```
from rake_nltk import Rake
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import
cosine_similarity
from sklearn.feature_extraction.text
import CountVectorizer
df = pd.read_csv('IMDB_Top250Engmovies2_OMDB_
_Detailed.csv')
df.head()
```

```
df['Director'].value_counts()[0:10].plo
t('barh', figsize=[8,5], fontsize=15,
color='navy').invert_yaxis()
```

- Название
- Режиссер
- Актеры
- Сюжет
- Жанр



# Подготовка данных

- удаление стоп-слов
- удаление пунктуации
- удаление пробелов
- конвертации всех слов в нижний регистр

```
df['Key_words'] = ''
r = Rake()
for index, row in df.iterrows():
    r.extract_keywords_from_text(row['Plot'])
    key_words_dict_scores = r.get_word_degrees()
    row['Key_words'] =
list(key_words_dict_scores.keys())
```

## Использование функции Rake для извлечения ключевых слов

Plot

Two imprisoned men bond over a number of years, finding ...

The aging patriarch of an organized crime dynasty transf...

The early life and career of Vito Corleone in 1920s New ...

When the menace known as the Joker emerges from his myst...

A jury holdout attempts to prevent a miscarriage of just...



Key\_words

[finding, solace, years, acts, number, eventual, redempt...

[aging, patriarch, organized, crime, dynasty, transfers,...

[portrayed, 1920s, new, york, family, crime, syndicate, ...

[mysterious, past, gotham, ability, dark, knight, must, ...

[miscarriage, jury, holdout, attempts, colleagues, just...




ОБРАЗОВАТЕЛЬНЫЙ  
ЦЕНТР  
МГТУ им. Н. Э. Баумана

# Подготовка данных

```
df['Genre'] = df['Genre'].map(lambda x: x.split(','))
df['Actors'] = df['Actors'].map(lambda x: x.split(',')[:3])
df['Director'] = df['Director'].map(lambda x: x.split(','))
for index, row in df.iterrows():
    row['Genre'] = [x.lower().replace(' ', '') for x in row['Genre']]
    row['Actors'] = [x.lower().replace(' ', '') for x in row['Actors']]
    row['Director'] = [x.lower().replace(' ', '') for x in row['Director']]
```

Director	Actors	Genre
Frank Darabont	Tim Robbins, Morgan Freeman, Bob Gunton, William Sadler	Crime, Drama
Francis Ford Coppola	Marlon Brando, Al Pacino, James Caan, Richard S. Castellano	Crime, Drama
Francis Ford Coppola	Al Pacino, Robert Duvall, Diane Keaton, Robert De Niro	Crime, Drama
Christopher Nolan	Christian Bale, Heath Ledger, Aaron Eckhart, Michael Caine	Action, Crime, Drama
Sidney Lumet	Martin Balsam, John Fiedler, Lee J. Cobb, E.G. Marshall	Crime, Drama



Director	Actors	Genre
[frankdarabont]	[timrobbins, morganfreeman, bobgunton]	[crime, drama]
[francisfordcoppola]	[marlonbrando, alpacino, jamescaan]	[crime, drama]
[francisfordcoppola]	[alpacino, robertduvall, dianekeaton]	[crime, drama]
[christophernolan]	[christianbale, heathledger, aaroneckhart]	[action, crime, drama]
[sidneylumet]	[martinbalsam, johnfiedler, lee.j.cobb]	[crime, drama]

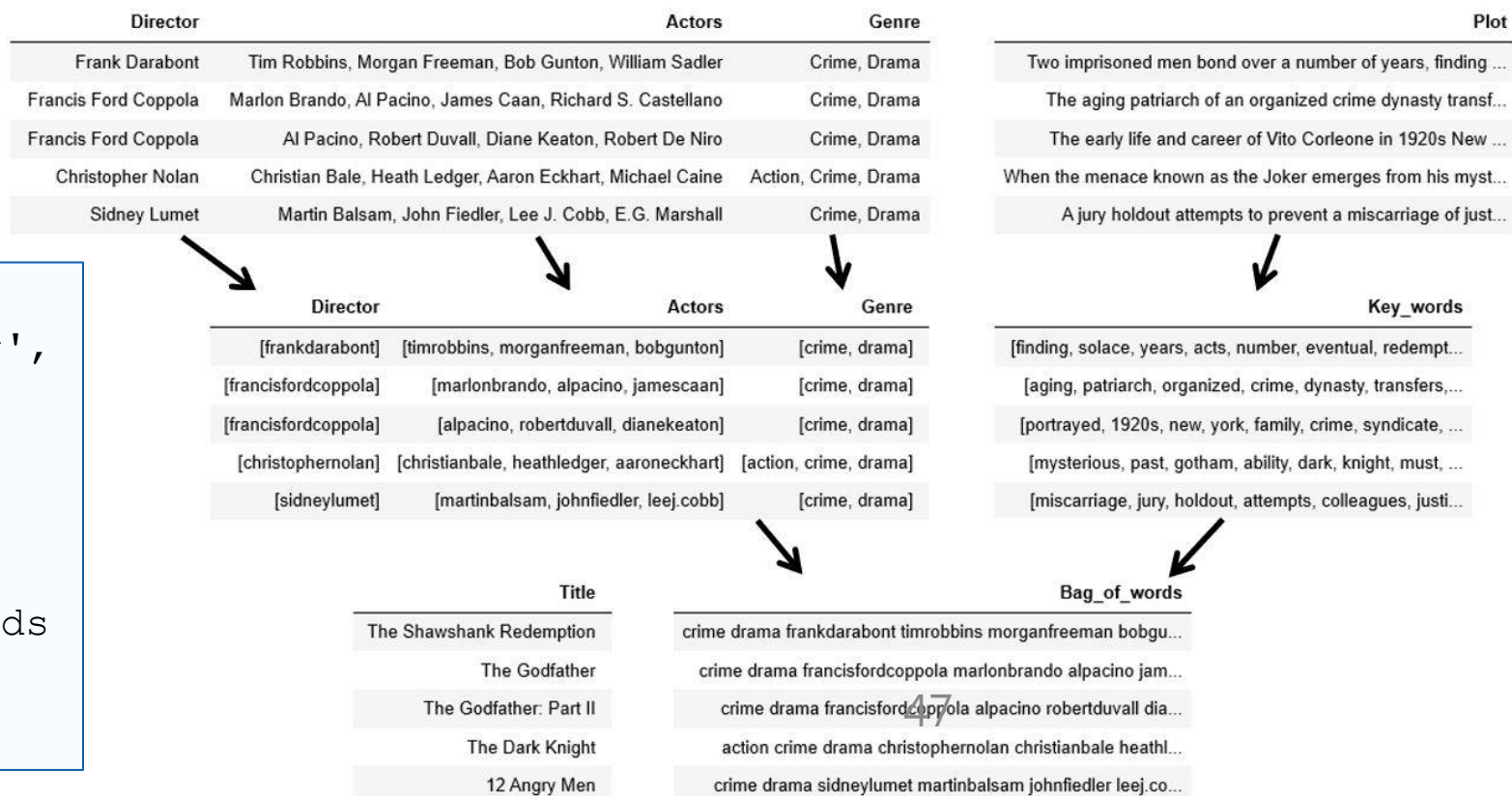
Все имена трансформированы в  
уникальные идентификационные  
значения

46

# Подготовка данных

```
df['Bag_of_words'] = ''
columns = ['Genre', 'Director',
           'Actors', 'Key_words']
for index, row in df.iterrows():
    words = ''
    for col in columns:
        words += ' ' + row[col]
    row['Bag_of_words'] = words

df = df[['Title', 'Bag_of_words']]
```





# Подготовка данных

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|} = \frac{\sum_{i=1}^n u_i v_i}{\sqrt{\sum_{i=1}^n u_i^2} \sqrt{\sum_{i=1}^n v_i^2}}$$

$$\mathbf{u} \cdot \mathbf{v} = [u_1 \ u_2 \ \dots \ u_n] \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = u_1 v_1 + u_2 v_2 + \dots + u_n v_n = \sum_{i=1}^n u_i v_i$$

Формула косинусного коэффициента для расчета значений в подобных матрицах

```
count = CountVectorizer()  
count_matrix = count.fit_transform(df['Bag_of_words'])  
cosine_sim = cosine_similarity(count_matrix, count_matrix)  
print(cosine_sim)
```

48



# Подготовка данных

	Movie0	Movie1	Movie2	...	Movie247	Movie248	Movie249
Movie0	[1.	0.15789474	0.13764944	...	0.05263158	0.05263158	0.05564149]
Movie1	[0.15789474	1.	0.36706517	...	0.05263158	0.05263158	0.05564149]
Movie2	[0.13764944	0.36706517	1.	...	0.04588315	0.04588315	0.04850713]
...	...						
Movie247	[0.05263158	0.05263158	0.04588315	...	1.	0.05263158	0.05564149]
Movie248	[0.05263158	0.05263158	0.04588315	...	0.05263158	1.	0.05564149]
Movie249	[0.05564149	0.05564149	0.04850713	...	0.05564149	0.05564149	1.]

Подобная матрица  
250 строк x 250 столбцов

Создать серию названий фильмов для того, чтобы список из названий совпадал со списком строк и столбцов матрицы подобия.

<sup>49</sup>  
`indices = pd.Series(df['Title'])`



# Обучение алгоритма

```
def recommend(title, cosine_sim = cosine_sim):  
    recommended_movies = []  
    idx = indices[indices == title].index[0]  
    score_series = pd.Series(cosine_sim[idx]).sort_values(ascending = False)  
    top_10_indices = list(score_series.iloc[1:11].index)  
    for i in top_10_indices:  
        recommended_movies.append(list(df['Title'])[i])  
    return recommended_movies
```

recommend('The Avengers')

```
['Guardians of the Galaxy Vol. 2',  
 'Aliens',  
 'Guardians of the Galaxy',  
 'The Martian',  
 'Interstellar',  
 'Blade Runner',  
 'Terminator 2: Judgment Day',  
 'The Thing',  
 'The Terminator',  
 'Spider-Man: Homecoming']
```



# Проблема холодного старта

**Холодный старт** (англ. cold start) — ситуация, когда еще не накоплено достаточное количество данных для корректной работы рекомендательной системы.

Как рекомендовать товар, который еще никто не видел?

Что рекомендовать пользователю, у которого еще нет ни одной оценки?

- Новый пользователь ещё ничего не оценил или оценил очень мало, чтобы сделать точные рекомендации.
- Сколько рейтингов нового продукта нужно, прежде чем его можно будет уверенно рекомендовать?
- Рейтинги искусственно корректируют.
- Новому пользователю скорее всего будет предлагаться самые популярные продукты — производители еще не раскрученного контента будут в убытке.

# Сингулярное разложение матрицы

## (Singular Value Decomposition, SVD)

- Коллаборативная фильтрация, content based фильтрация и similarity based фильтрация имеют ряд схожих проблем:
  - Проблема холодного старта.
  - Плохие предсказания для новых/ нетипичных пользователей/объектов.
  - Тривиальность рекомендаций.
  - Ресурсоемкость вычислений. Для того, чтобы делать предсказания нам нужно держать в памяти все оценки всех пользователей.

### SVD

- Показывает геометрическую структуру матрицы и позволяет наглядно представить имеющиеся данные.
- Где используется: от приближения методом наименьших квадратов и решения систем уравнений до сжатия изображений.

# Алгоритм SVD

$$R_{n \times m} = U_{n \times n} \times \Sigma_{n \times m} \times V_{m \times m}^T$$

усеченное разложение

$$R'_{n \times m} = U'_{n \times d} \times \Sigma'_{d \times d} \times V_{d \times m}'^T$$

$$R'_{n \times m}$$

наилучшее низкоранговое приближение  
с точки зрения средне-квадратичного отклонения

Упростим

$$\tilde{U}_{n \times d} = U'_{n \times d} \times \Sigma'_{d \times d}$$

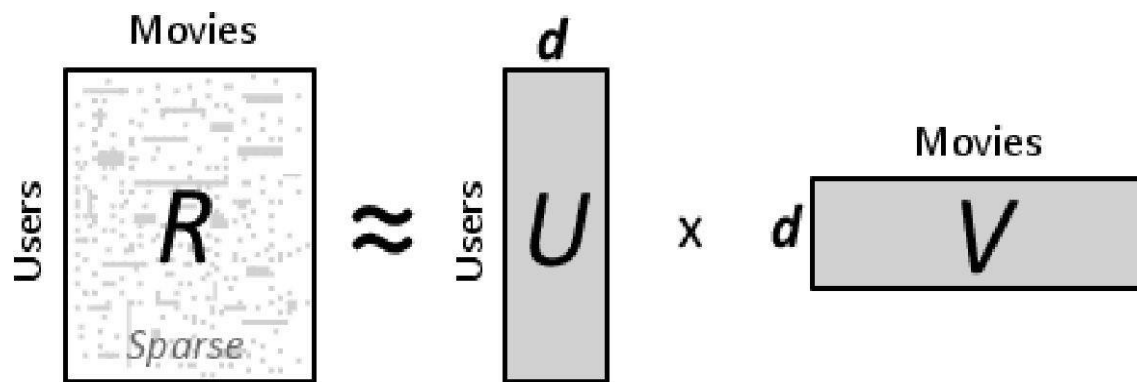
$$V_{d \times m}'^T \longrightarrow \tilde{V}_{d \times m}$$

$$R'_{n \times m} = \tilde{U}_{n \times d} \times \tilde{V}_{d \times m}$$

приближенная матрица оценок может быть вычислена как  
произведение усеченных матриц пользователей и оценок

# Алгоритм SVD

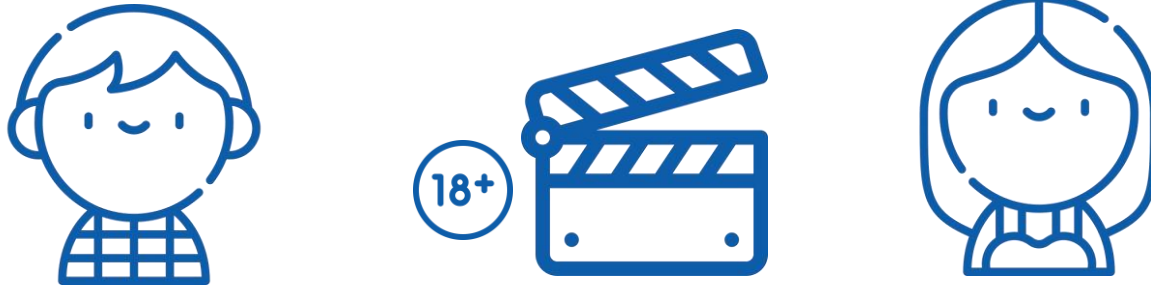
- Чтобы предсказать оценку пользователя  $U$  для объекта  $i$ , берём некоторый вектор  $p_u$  (набор параметров) для данного пользователя и вектор данного объекта  $q_i$ .
- Их скалярное произведение – и есть нужное нам предсказание:
- $\widehat{r_{ui}} = \langle p_u, q_i \rangle$



$$\widehat{r_{ui}} = \langle p_u, q_i \rangle$$

# Сингулярное разложение матрицы (SVD)

- Позволяет выявлять скрытые признаки объектов и интересы пользователей.



- Матрица оценок  $R$  полностью не известна, поэтому просто взять SVD разложение не представляется возможным;
- Сингулярное разложение не единственное, поэтому даже если какое-то разложение будет найдено, нет гарантии, что первая координата в нем будет соответствовать некоторым выбранным характеристикам пользователя.

# Таким образом:

- Оффлайн-системы (персонализированные):
  - метод ближайших соседей: GroupLens;
  - SVD-разложение матриц градиентным спуском, его байесовская версия (PMF);
  - машины Больцмана – модель пользователя;
  - их расширения: время, контекст, холодный старт.
- Онлайн-системы (поиск трендов):
  - модель DGP для предсказания популярности;
  - многорукие бандиты для решения explore-exploit;
  - расширения:
    - многокритериальная оптимизация,
    - частичная персонализация.