Рекомендательные системы

Цель занятия

После освоения темы:

- вы узнаете, что такое рекомендательные системы;
- познакомитесь с видами рекомендательных систем;
- получите базовые сведения о рекомендательных системах.

План занятия

1. Понятие рекомендательных систем.
2. Виды рекомендательных систем и их алгоритмы.
3. Проблема холодного старта.
4. Актуальность рекомендаций.
5. Стандартизация данных.
6. Обоснование результатов.
7. Итоги.
8. Подборка полезных ресурсов.

Конспект занятия

Понятие рекомендательных систем

Рекомендательные системы — это программы, которые анализируют предпочтения пользователей и предсказывают, что может понравиться им в будущем. Их алгоритмы часто построены на основе машинного обучения: модель учится на выборе пользователя и предлагает ему новые возможности взаимодействия.

Рекомендации используют в разных бизнесах:

- Интернет-магазины и витрины предлагают выбрать товар в разделах «Популярное за месяц», «С этим товаром также покупают» и «Вам может понравиться».
- Музыкальные и видеостриминги собирают плейлист дня или советуют фильм на вечер.
- Медиа показывают материалы, которые могут понравиться пользователю.
- Социальные сети предлагают добавить в контакты новых друзей.

Задача рекомендательной системы — проинформировать пользователя о товаре, который ему может быть наиболее интересен в данный момент времени.

Где можно встретить рекомендательные системы? С рекомендательными системами можно столкнуться там, где есть большое множество товаров и пользователей, которые хотят найти нужные для себя товары. Рекомендательные системы помогают отобрать наиболее релевантные для пользователя объекты, тем самым экономя его время.

Примеры:

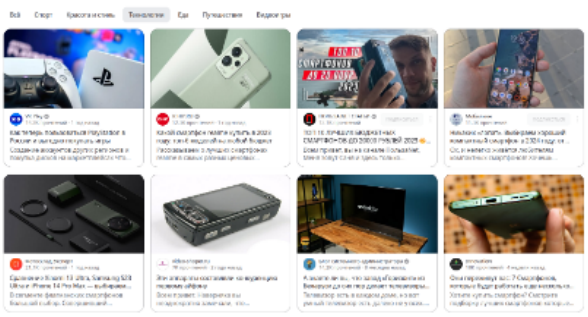
➔ YouTube рекомендует пользователям видео.

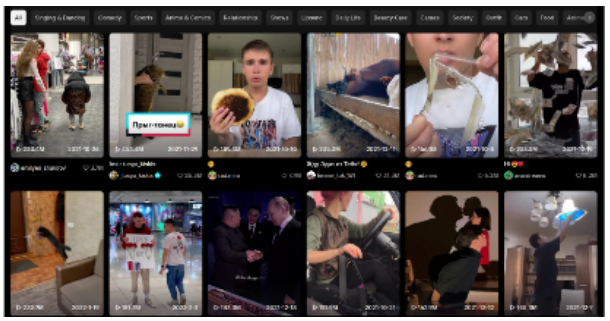
➔ На сайтах интернет-магазинов можно встретить блоки с рекомендациями товаров.

➔ Музыкальные сервисы наподобие Spotify или Яндекс.Музыки рекомендуют музыкальные треки.

➔ Блог-платформы (Дзен) рекомендуют интересный контент для пользователей.

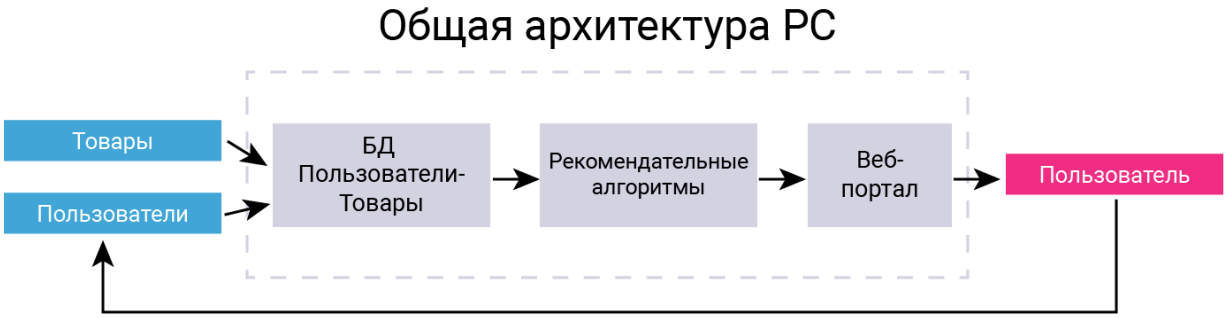
➔ Платформа для просмотра коротких видеороликов (TikTok, YouTube shorts и др.) предоставляет пользователю рекомендованный контент.





Что из себя представляют «релевантные для пользователя товары» — это нетривиальный вопрос, который решается отдельно для каждой задачи исходя из бизнес-логики.

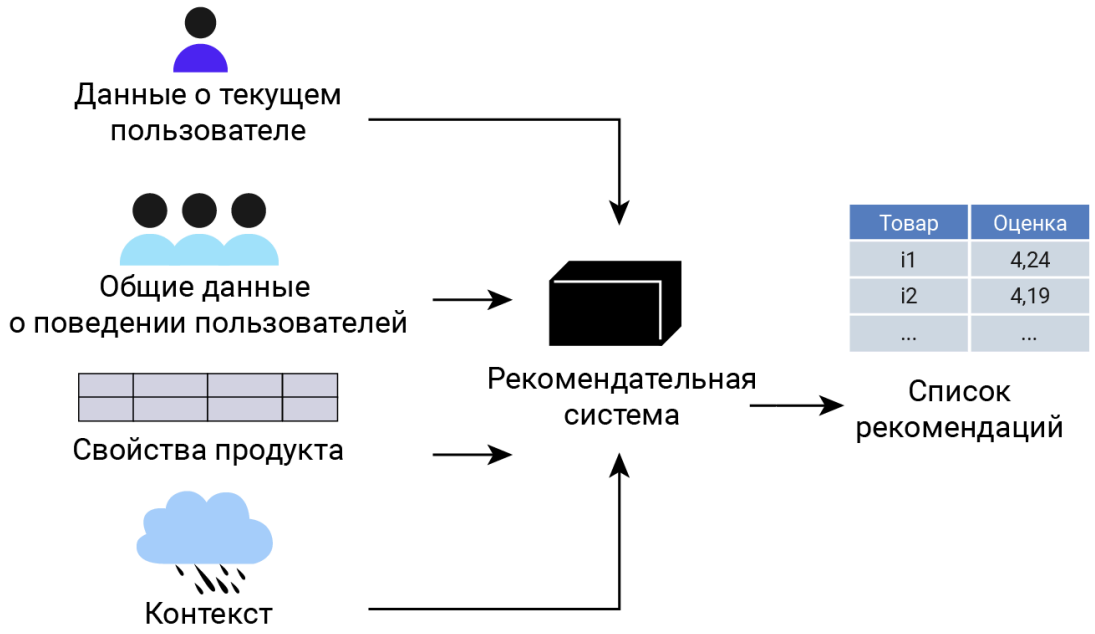
Виды рекомендательных систем и их алгоритмы



Рекомендательной системой принято называть набор алгоритмов, сервисов или других программных продуктов, позволяющих определить предпочтения конкретного человека. Для этого используются различные данные — от личной информации о пользователе до сделанных им запросов в поисковых системах. Результатом становится выведение на экран именно тех предложений, которые окажутся интересными ему.

Виды рекомендательных систем:

- ➔ Фильтрация, основанная на контенте (content-based filtering).
- ➔ Коллаборативная фильтрация (collaborative filtering).
- ➔ Гибридные рекомендательные системы (hybrid filtering).



Данные для рекомендательных систем

Принцип работы любой рекомендательной системы предельно прост. Она базируется на алгоритмах искусственного интеллекта, главной целью которых становится сбор и анализ информации о потенциальном клиенте. Источником данных становится все: поисковые запросы, время просмотра ролика на YouTube, личные сведения из профиля в социальной сети, сделанные и несделанные покупки и др.

В центре любой рекомендательной системы находится так называемая матрица предпочтений. Это матрица, по одной из осей которой отложены все клиенты сервиса (users), а по другой — объекты рекомендации (items). На пересечении некоторых пар (user, item) данная матрица заполнена оценками (ratings) — это известный нам показатель заинтересованности пользователя в данном товаре, выраженный по заданной шкале (например, от 1 до 5).

	Товар 1	Товар 2	Товар 3	Товар 4	Товар 5
Клиент 1		3		5	
Клиент 2	1		1	1	
Клиент 3	2			3	2
Клиент 4		4			5
Клиент 5	5		2	3	4

Пользователи обычно оценивают лишь небольшую часть товаров, что есть в каталоге, и задача рекомендательной системы — обобщить эту информацию и предсказать отношение клиента к другим товарам, про которые ничего не известно. Другими словами, нужно заполнить все незаполненные ячейки на картинке выше.

Шаблоны потребления у людей разные, и не обязательно должны рекомендоваться новые товары. Можно показывать повторные позиции — например, для пополнения запаса.

По этому принципу выделяют **две группы товаров**:

- Повторяемые (шампуни или туалетная бумага, которые нужны всегда).

- Неповторяемые (книги или фильмы, которые редко приобретают повторно).

Если продукт нельзя явно отнести к одному из классов, имеет смысл определять допустимость повторных покупок индивидуально (кто-то ходит в магазин только ради шоколада определенной марки, а кому-то важно попробовать все, что есть в каталоге).

Понятие «интересности» тоже субъективное. Некоторым пользователям нужны вещи только из их любимой категории, а кто-то, наоборот, больше откликается на нестандартные товары или группы товаров. Например, видеохостинг может рекомендовать пользователю только новые серии любимого сериала, а может

периодически закидывать ему новые шоу или вообще новые жанры.
В идеале стоит выбирать стратегию показа рекомендаций под каждого клиента отдельно, с помощью моделирования категории клиента.

Способы получения пользовательских оценок:

- Явно (explicit ratings) — пользователь сам ставит рейтинг товару, оставляет отзыв, лайкает страницу.

- Неявно (implicit ratings) — пользователь явно свое отношение не выражает, но можно сделать косвенный вывод из его действий: купил товар — значит, он ему нравится; долго читал описание — значит, есть интерес и т. п.

Конечно, явные предпочтения лучше: пользователь сам говорит о том, что ему понравилось. Однако на практике далеко не все сайты предоставляют возможность явно выражать свой интерес, да и не все пользователи имеют желание это делать. Чаще всего используются сразу оба типа оценок, которые хорошо дополняют друг друга.

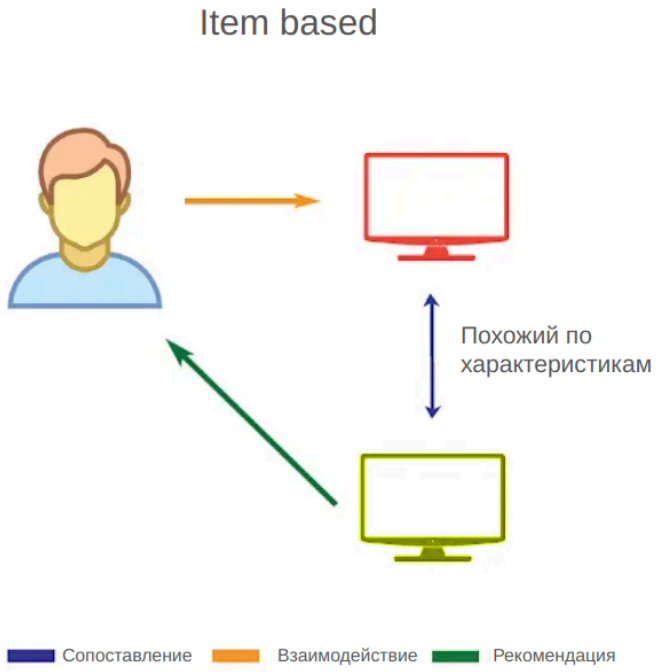
Фильтрация на основании контента (content-based filtering)

Рекомендации content-based предполагают максимальное использование информации о самом пользователе или товаре. Это простой и очевидный способ определения предпочтений и интересов пользователя.

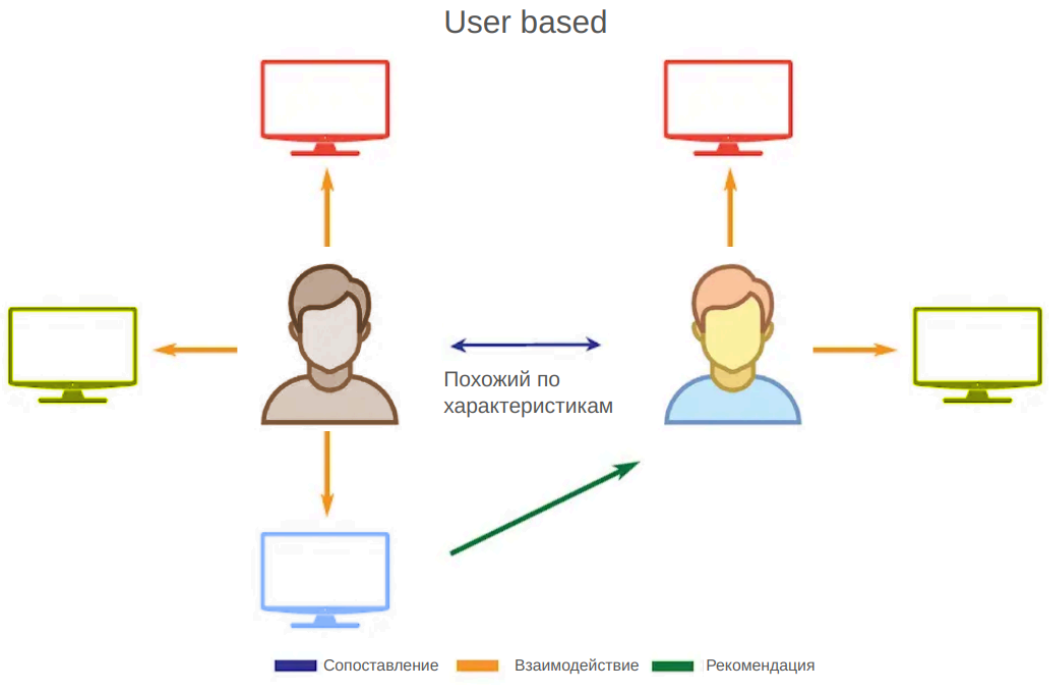
Выделяют два вида фильтрации, основанной на контенте, — User Based и Item Based.

Item Based использует информацию о характеристиках товара для рекомендаций. Если человек посмотрел боевик, будет вполне логичным предложить ему еще несколько фильмов аналогичного жанра. Прослушивание трека рок-исполнителя наверняка означает, что программа порекомендует как еще несколько песен от него, так и произведения других авторов этого направления музыки.

Плюс такого подхода к фильтрации очевиден: он требует минимума ресурсов и предельно прост. Но при этом имеется и явный минус. Например, если клиент совершил дорогую покупку (автомобиль), не имеет смысла предлагать ему еще 10 машин: вероятность еще одного такого приобретения крайне мала. В этом случае помогает другой вид рекомендательных систем.



User Based использует информацию о характеристиках пользователей для рекомендаций. Если пользователь А и пользователь Б имеют похожие характеристики (похожий возраст, устройство, местоположение, схожие вкусы и предпочтения и др.), то система может рекомендовать пользователю А тот контент, с которым пользователь А не взаимодействовал, но с которым взаимодействовал пользователь Б.



Очевидное требование для систем рекомендаций, основанных на контенте, — у всех товаров/пользователей должно быть описание.

Исторически предметом рекомендаций content-based чаще были товары с неструктурированным описанием: фильмы, книги, статьи. Такими признаками могут быть, например, текстовые описания, рецензии, состав актеров и прочее. Однако ничто не мешает использовать и обычные числовые или категориальные признаки.

Неструктурированные признаки описываются типичным для текста способом — векторами в пространстве слов. Каждая размерность такого вектора — это признак, потенциально характеризующий интерес пользователя.

По мере взаимодействия пользователя с системой (скажем, он покупает фильмы) векторные описания приобретенных им товаров объединяются (суммируются и нормализуются) в единый вектор; и таким образом формируется вектор его интересов. Далее достаточно найти товар, описание которого наиболее близко к вектору интересов, т. е. решить задачу поиска n ближайших соседей.

Не все элементы одинаково значимы: например, союзные слова, очевидно, не несут никакой полезной нагрузки. Поэтому при определении числа совпадающих элементов в двух векторах все измерения нужно предварительно взвешивать по их значимости. Данную задачу решает преобразование TF-IDF, которое назначает больший вес более редким интересам. Совпадение таких интересов имеет большее значение при определении близости двух векторов, чем совпадение популярных.

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Вес слова x в описании товара y

$tf_{x,y}$ = частота слова x в описании товара y

df_x = количество товаров, содержащих слово x

N = общее количество товаров

Принцип TF-IDF здесь в той же мере применим и к обычным номинальным атрибутам, таким как жанр, режиссер, язык. TF — мера значимости атрибута для пользователя, IDF — мера «редкости» атрибута.

Некоторые моменты, которые можно учесть при реализации:

- При формировании vector-space представления товара вместо отдельных слов можно использовать n-граммы (последовательные пары слов, тройки и т. д.). Это сделает модель более детализированной, однако потребуется больше данных для обучения.

- В разных местах описания товара вес ключевых слов может отличаться. Например, описание фильма может состоять из заголовка, краткого описания и детального описания.

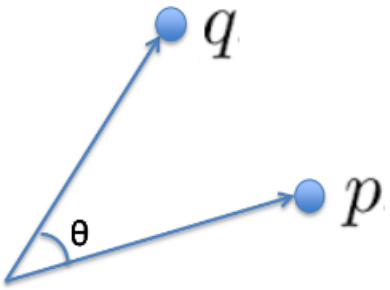
- Описания товара от разных пользователей можно взвешивать по-разному. Например, можем давать больший вес активным пользователям, у которых много оценок.

- Аналогично можно взвешивать и по товару: чем больше средний рейтинг объекта, тем больше его вес.

- Если описание товара допускает ссылки на внешние источники, то можно заморочиться и анализировать также всю связанную с товаром стороннюю информацию.

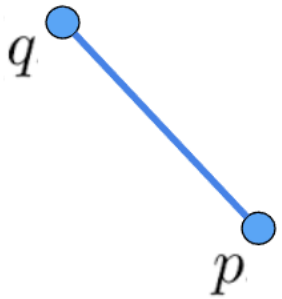
Чтобы рассчитать расстояние между двумя векторами, чаще всего используют косинусное расстояние:

$$d(p, q) = 1 - \cos(\theta)$$



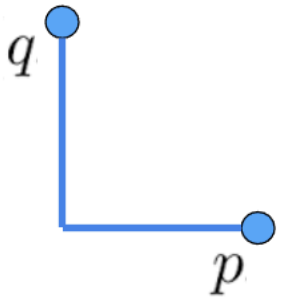
Также можно использовать евклидово расстояние:

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$



Или же манхэттенское расстояние:

$$d(p,q) = \sum_{i=1}^{n} \left| q_i - p_i \right|$$



Или же можно посчитать похожесть двух векторов.

Перечислим несколько наиболее популярных:

1. **Корреляция Пирсона** — классический коэффициент, который вполне применим и при сравнении векторов:

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}$$

2. **Корреляция Спирмана.** Основное отличие — коэффициент ранговый, т. е. работает не с абсолютными значениями рейтингов, а с их порядковыми номерами. В целом дает результат, очень близкий к корреляции Пирсона.

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

3. **Косинусная похожесть.** Если два вектора сонаправлены (т. е. угол между ними нулевой), то косинус угла между ними равен единице. И наоборот, косинус угла между перпендикулярными векторами равен нулю.

$$sim(A,B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

Пример фильтрации content-based через поиск ближних соседей

Будем работать с данными с каггл про рекомендацию книг.
Считаем данные про книги, удаляем пропуски в описании и оставляем необходимые столбцы:

```
books = pd.read_csv('book1000k-1100k.csv')
books = books.dropna(subset='Description')
books = books[['Name', 'Authors', 'Description']]
books.head()
```

	Name	Authors	Description
0	Flight from Eden	Kathryn A. Graham	What could a computer expert, a mercenary with...
1	Roommates Again	Kathryn O. Galbraith	During their stay at Camp Sleep-Away, sisters ...
2	The King At The Door	Brock Cole	A poorly dressed old man appears at an inn and...
3	Giotto: The Scrovegni Chapel, Padua	Bruce Cole	This beautiful series lavishly illustrates the...
4	Larky Mavis	Brock Cole	Another orginal picture-book fairy tale...

Сразу же считаем таблицу с пользователями, переводим текстовый рейтинг в численный, удаляем пользователей без рейтингов и соедининяем с таблицей с описанием книг:

```
users = pd.read_csv('user_rating_0_to_1000.csv')
map_rating = {
      'it was amazing': 5,
      'really liked it': 4,
      'liked it': 3,
      'it was ok': 2,
      'did not like it': 1,
      "This user doesn't have any rating": 0
}
users['rating'] = users['Rating'].replace(map_rating)
users = users[~(users['rating'] == 0)]

ratings = users.merge(books, on='Name')
ratings.head()
```

	ID	Name	Rating	rating	Authors	Description
0	1	Siddhartha	it was amazing	5	Hermann Hesse	This classic of twentieth-century literature c...
1	1	The Authoritative Calvin and Hobbes: A Calvin ...	it was amazing	5	Bill Watterson	The Authoritative Calvin and Hobbes, is...
2	1	The Return of the King (The Lord of the Rings,...	it was amazing	5	J.R.R. Tolkien	The third volume in J.R.R. Tolkien's epic adve...
3	1	Freakonomics: A Rogue Economist Explores the H...	really liked it	4	Steven D. Levitt	Steven D. Levitt and Stephen J. Dubner...
4	1	The Golden Compass (His Dark Materials, #1)	it was amazing	5	Philip Pullman	Lyra's life is already sufficiently interestin...

Оставим только уникальные книги, на их описании построим векторизатор Tfldf, преобразуем все описания книг и на них обучим модель Nearest Neighbors для поиска ближайших соседей по признакам:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import NearestNeighbors


unique_books = ratings.drop_duplicates('Name')

tfidf = TfidfVectorizer(max_features=10000, stop_words='english')
books_tfidf = tfidf.fit_transform(unique_books['Description'])


neigh = NearestNeighbors(n_neighbors=7, metric='euclidean')
neigh.fit(books_tfidf)
```

Дело остается за малым: взять нужного пользователя и его любимую книгу и на ней сделать предсказания наиболее похожих книг. Для этого возьмем книгу «Маленький принц», найдем ее описание и прогоним через Tfldf. На полученном векторе найдем ближайших соседей:

```


test_movie = "The Little Prince"
test_description = ratings[ratings['Name'] ==
test_movie]['Description'].iloc[0]

X_tfidf = tfidf.transform([test_description])

res = neigh.kneighbors(X_tfidf, return_distance=True)

res
```

```
(array([[0.00000002, 1.        , 1.33846399, 1.34587702, 1.35087489,
         1.35114525, 1.35326698]]),
 array([[ 38, 807, 102, 863, 692,  84, 348]]))
```

В результате получаем расстояния до ближайших соседей и их индексы. По ним-то и получаем названия рекомендованных книг:

```
rec_names = unique_books.iloc[res[1][0]]['Name']
rec_names
```

```
38                                  The Little Prince
2852               To the Nines (Stephanie Plum, #9)
120     Little Town on the Prairie (Little House, #7)
3367                               The Art of Mending
2143          The First Four Years (Little House, #9)
94                                      Goodnight Moon
841                          Where Are the Children?
Name: Name, dtype: object
```

На первом месте получаем целевую книгу, от которой искали соседей: конечно же, ее повторно рекомендовать не будем. А остальные можем показать пользователю, так как по описанию они очень похожи на его любимую книгу.

Коллаборативная фильтрация (collaborative filtering)

Коллаборативная система рекомендаций анализирует данные о предпочтениях и действиях пользователей, чтобы предложить им подходящий контент. Если пользователь совершал покупки или же просто переходил на страницу товара, то система находит других пользователей с похожими запросами. После этого система рекомендует пользователю те продукты, которыми интересовались другие клиенты, а целевой пользователь — еще нет.

User-based
Подход описывает пользователей их взаимодействиями с товарами:

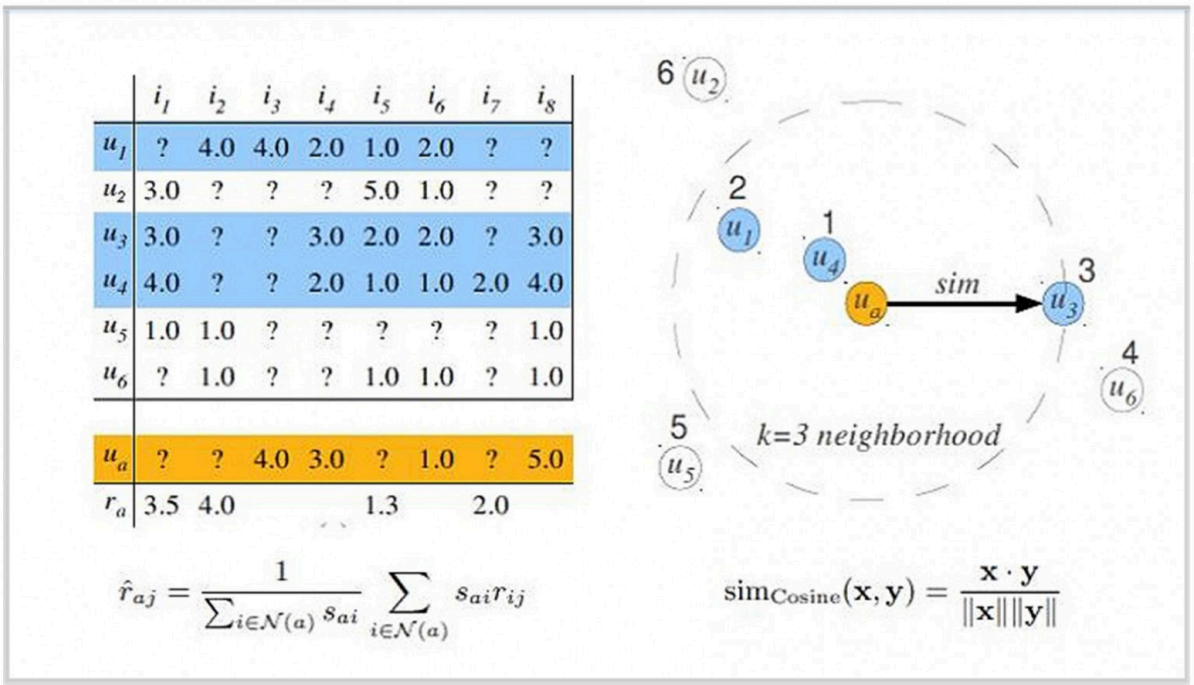
	item_1	item_2	…	item_m
user_1	5	?	…	3
user_2	?	2		5
…	…	…	…	…
user_n	2	2	…	?

Item-based

Подход описывает товары через взаимодействия с пользователями:

	user_1	user_2	…	user_n
item_1	5	?	…	2
item_2	?	2		2
…	…	…	…	…
item_m	3	5	…	?

Классическая реализация алгоритма основана на принципе k-ближайших соседей, где для каждого пользователя ищем k наиболее похожих на него (в терминах предпочтений) и дополняем информацию о пользователе известными данными по его соседям. Так, если известно, что ваши соседи по интересам в восторге от фильма «Унесенные ветром», а вы его по какой-то причине еще не смотрели, это отличный повод предложить вам данный фильм для субботнего просмотра.



На изображении выше проиллюстрирован принцип работы метода. В матрице предпочтений желтым цветом выделен пользователь, для которого мы хотим определить оценки по новым товарам (знаки вопроса). Синим цветом выделены три его ближайших соседа.

«Похожесть» в данном случае — синоним «корреляции» интересов, и может считаться множеством способов (помимо корреляции Пирсона есть еще косинусное расстояние, расстояние Жаккара, расстояние Хэмминга и пр.).

У классической реализации алгоритма есть один явный минус: он плохо применим на практике из-за квадратичной сложности. Действительно, как любой метод ближайшего соседа, он требует расчета всех попарных расстояний между пользователями (а пользователей могут быть миллионы).

Данная проблема отчасти может быть решена, если ввести корректировки в алгоритм:

- обновлять расстояния не при каждой покупке, а батчами (например, раз в день);
- не пересчитывать матрицу расстояний полностью, а обновлять ее инкрементально;
- сделать выбор в пользу итеративных и приближенных алгоритмов (например, ALS).

Пример коллаборативной фильтрации

Продолжаем работать с данными с каггл про рекомендацию книг. Возьмем только тех пользователей, у которых есть минимум 5 проставленных рейтингов, и на них построим сводную таблицу:

```
users_inds = ratings['ID'].value_counts()[ratings['ID'].value_counts() >
5].index
ratings = ratings[ratings['ID'].isin(users_inds)]

user_based = ratings.pivot_table(index='ID', columns='Name',
values='rating').fillna(0)
user_based
```

Name ID	1000 Record Covers	1st to Die (Women's Murder Club, #1)	20,000 Leagues Under the Sea	4:50 from Paddington (Miss Marple, #8)	4th of July (Women's Murder Club, #4)	A Beautiful Blue Death (Charles Lenox Mysteries, #1)	A Bed of Red Flowers: In Search of My Afghanistan	A Book of Common Prayer	A Boy's Own Story (The Edmund Trilogy, #1)	A Briefer History of Time	...
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
...	...	...	...	...	...	...	...	...	...	...	...

Затем выберем целевого пользователя, для которого хотим сделать рекомендации, получим для него вектор взаимодействия с товарами. Пройдемся в цикле по всей матрице взаимодействия, каждый раз получая вектор взаимодействия пользователей с товарами, и будем считать косинусное расстоянием между целевым пользователем и остальными из датасета:

```
from scipy.spatial.distance import cosine

ind = 103
target_vector = user_based.loc[ind]

users_ind = []
distances = []

for i, row in user_based.iterrows():
      if i == ind:
      continue

      users_ind.append(i)
      distances.append(cosine(row, target_vector))

len(distances)
```

Отберем топ-10 пользователей, на которых целевой больше всех похож, и напечатаем их идентификаторы и расстояние до них:

```
best_indexes = np.argsort(distances)[:10]
best_users = [(users_ind[i], distances[i]) for i in best_indexes]

for m in best_users:
```

```
    print(m)
```

```
(377, 0.703681121005123)
(169, 0.7098094999559953)
(535, 0.7612887710067714)
(136, 0.763902699117777)
(148, 0.7648205950386266)
(507, 0.7718729344379365)
(694, 0.7827211836519412)
(550, 0.7888429456842572)
(503, 0.7923319565745307)
(655, 0.79238630036565)
```

Давайте возьмем только одного пользователя, на которого больше всего похож целевой, и посмотрим, какие книги он высоко оценил:

```
closest_user_ratings = user_based.loc[best_users[0][0]]
closest_user_books = set(closest_user_ratings[closest_user_ratings >=
3].index)
closest_user_books
```

```
{'Animal Farm',
 'Jonathan Livingston Seagull',
 'Love in the Time of Cholera',
 'Middlesex',
 'Possession',
 'Siddhartha',
 'The Collector',
 'The Heart of the Matter',
 'The Little Prince',
 'The Plague',
 'The Sound and the Fury',
 'The Trial'}
```

Также найдем книги, которые пользователь уже читал:

```
target_user_books = set(target_vector[target_vector != 0].index)
target_user_books
```

```
{'A Clockwork Orange',
 'Blink: The Power of Thinking Without Thinking',
 "Bridget Jones's Diary",
 'Healthy Sleep Habits, Happy Child',
 'Illuminati (Robert Langdon, #1)',
 'Middlesex',
 'Siddhartha',
 'Summer People',
 'The Beach',
 'The Little Prince',
 'The Lovely Bones',
 'The War of the Worlds',
 'Water for Elephants'}
```

Остается найти разницу двух множеств:

```
closest_user_books - target_user_books
```

```
{'Animal Farm',
 'Jonathan Livingston Seagull',
 'Love in the Time of Cholera',
 'Possession',
 'The Collector',
 'The Heart of the Matter',
 'The Plague',
 'The Sound and the Fury',
 'The Trial'}
```

Эти книги и будем рекомендовать нашему целевому пользователю.

Гибридные системы

Каждая из описанных выше рекомендательных систем имеет как ярко выраженные плюсы, так и не менее существенные минусы. Именно поэтому наибольшее распространение получили гибридные фильтрации, представляющие собой комбинацию из разных способов выдачи рекомендаций.

При таком подходе к решению задачи особенно актуальным становится баланс между ними. Например, для рекомендаций фильмов для пользователя А, сначала можем посмотреть, какие фильмы предпочитает пользователь А, основываясь на характеристиках фильмов, которые он уже смотрел (**content-based**). А затем среди выбранных фильмов можно посмотреть, какие фильмы высоко оценивали похожие на А пользователи (**коллаборативная**).

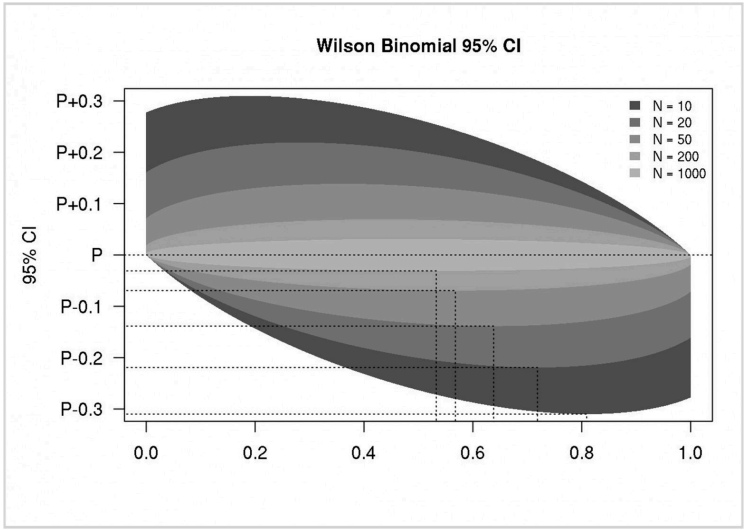
Проблема холодного старта

Холодный старт — это типичная ситуация, когда еще не накоплено достаточное количество данных для корректной работы рекомендательной системы (например, когда пользователь новый). Поэтому можно делать неперсонализированную систему рекомендаций, которая строится для среднего пользователя сервиса. Для этого нужно посчитать средний рейтинг товаров. Но если средний рейтинг посчитан по оценкам всего трех пользователей, такая оценка явно не будет достоверной. Часто в таких ситуациях рейтинги искусственно корректируют.

Первый способ — показывать не среднее значение, а сглаженное среднее. Смысл таков: при малом количестве оценок отображаемый рейтинг больше тяготеет к некому безопасному «среднему» показателю, а как только набирается достаточное количество новых оценок, «усредняющая» корректировка перестает действовать.

Второй способ — рассчитывать по каждому рейтингу интервалы достоверности. Математически, чем больше оценок, тем меньше вариация среднего и, значит, больше уверенность в его корректности. А в качестве рейтинга можно выводить, например, нижнюю границу интервала. При этом понятно, что такая система будет достаточно консервативной, с тенденцией к занижению оценок по новым товарам.

Поскольку оценки ограничены определенной шкалой (например, от 0 до 1), обычный способ расчета интервала достоверности здесь плохо применим: из-за хвостов распределения, уходящих на бесконечность, и симметричности самого интервала. Есть альтернативный и более точный способ его посчитать — Wilson Confidence Interval. При этом получаются несимметричные интервалы примерно такого вида:



На иллюстрации выше по горизонтали отложена оценка среднего значения рейтинга, по вертикали — разброс вокруг среднего значения. Цветом выделены различные размеры выборки (очевидно, чем выборка больше, тем меньше интервал достоверности).

Актуальность рекомендаций

В некоторых случаях также важно учитывать «свежесть» рекомендации. Это особенно актуально для статей или постов на форумах. Свежие записи должны чаще попадать в топ. Для этого используются корректирующие коэффициенты. Ниже пара формул для расчета рейтинга статей на медиасайтах.

Пример расчета рейтинга в журнале Hacker news:

$$Rank = \frac{(U-D-1)^{0.8} * P}{T^{1.8}}$$

где U=upvotes, D=downvotes, а P=penalty — дополнительная корректировка для имплементации иных бизнес-правил.

Расчет рейтинга в Reddit:

$$Rank = log_{10}(max(1, U-D)) - \frac{|U-D|T}{const}$$

где U=upvotes, D=downvotes, T=время записи. Первое слагаемое оценивает «качество записи», а второе делает поправку на время.

Очевидно, что универсальной формулы не существует, и каждый сервис изобретает ту формулу, которая лучше всего решает его задачу; проверяется это эмпирически.

Стандартизация данных (scaling)

Поскольку все пользователи оценивают по-разному (кто-то всем подряд пятерки ставит, а от кого-то четверки редко дождешься), перед расчетом данные лучше **нормализовать**, т. е. привести к единой шкале, чтобы алгоритм мог корректно сравнивать их между собой.

Естественно, предсказанную оценку затем нужно будет перевести в исходную шкалу обратным преобразованием (и, если нужно, округлить до ближайшего целого числа).

Нормализовать можно несколькими способами:

- Центрированием: из оценок пользователя просто вычитаем его среднюю оценку, что актуально только для небинарных матриц.

- Стандартизацией: в добавок к центрированию делим оценку ее на стандартное отклонение у пользователя. Здесь после обратного преобразования рейтинг может выйти за пределы шкалы (например, 6 по пятибалльной шкале), но такие ситуации довольно редки и решаются просто округлением в сторону ближайшей допустимой оценки.

- Двойной стандартизацией: первый раз нормируем оценками пользователя, второй раз — оценками товара. Если у фильма «А» средняя оценка 2.5, а пользователь ей ставит 5, то это сильный фактор, говорящий о том, что такие фильмы ему явно по вкусу.

Обоснования рекомендаций

Важно, чтобы пользователь доверял рекомендательной системе, а для этого она должна быть проста и понятна. При необходимости всегда должно быть доступно понятное объяснение рекомендации.

В рамках объяснения неплохо показывать оценку товара соседями, по какому именно атрибуту (например, актер или режиссер) было совпадение, а также выводить уверенность системы в оценке.

Например:

- «Вам может понравиться фильм…, поскольку там играет… и ...».
- «Пользователи с похожими на ваш музыкальными вкусами оценили альбом… на 4.5 из 5».

Итоги

1. Рекомендательная система стремится максимально точно предсказать предпочтения потребителя и предложить наиболее подходящий товар или услугу.

2. Сегодня такие системы встречаются повсеместно. Практически любой крупный интернет-магазин, онлайн-кинотеатр или новостной портал использует ту или иную рекомендательную систему для того, чтобы предоставить пользователям то, что им действительно нужно.

3. В системах рекомендаций работают с разными мнениями пользователя:

- явная обратная связь (explicit) — пользователь сам ставит рейтинг товару, оставляет отзыв, лайкает страницу;
- неявная обратная связь (implicit) — пользователь явно свое отношение не выражает. Покупка, клик, долгое нахождение на странице с товаром.

4. Есть несколько видов систем рекомендаций:

- Content-Based, которая строится на характеристиках товара/пользователя.
- Коллаборативная, которая берет во внимание взаимодействия пользователей и товаров.
- Гибридная. Для нее можем использовать несколько систем фильтрации вместе.

5. Стоит стандартизировать оценки пользователей, чтобы предсказывать более объективную величину.

Подборка полезных ресурсов

1. Анатомия рекомендательных систем. Часть первая
2. Введение в рекомендательные системы
3. Рекомендательная система | Вводный курс ML
4. Видео. Лекция. Машинное обучение. Рекомендательные системы. К. В. Воронцов, Школа анализа данных, Яндекс
5. Видео. Рекомендательные системы: архитектура и применение
6. Создание рекомендательных систем с использованием библиотеки Surprise
7. Рекомендательные системы: user-based и item-based