

Основы рекомендательных
систем.

Ассоциативные правила

Рекомендательные системы

Рекомендательные системы используются там, где множество пользователей выбирают объекты из большого множества объектов.

- U – множество пользователей.
- I – множество объектов.

Для каждого пользователя $u \in U$ есть множество объектов $I_u \subset I$, с которыми он взаимодействовал и которым поставил рейтинги $R_u = (r_{ui})_{i \in I_u}$.

- Рейтинг (фидбек) – это некоторая характеристика взаимодействия пользователя с объектом.
- Фидбек: explicit – явный (оценка, лайк/дизлайк, рецензия) и implicit – неявный (просмотр, клик, время).

Таким образом, задачу рекомендательных систем можно переформулировать в следующем виде: для каждого пользователя $u \in U$ необходимо оценить значение r_{ui} для $i \in I \setminus I_u \subset I$ и выбрать несколько объектов с наибольшим \hat{r}_{ui} .


















Задачу рекомендательной системы можно сформулировать как задачу регрессии или классификации. Однако во многих случаях достаточно использовать ранжирующую модель, которая не определяет точное значение рейтинга, а ранжирует объекты в порядке убывания рейтинга.

Коллаборативная фильтрация

Коллаборативная фильтрация – подход, объединяющий семейство методов рекомендаций, использующих сходство по истории взаимодействия между пользователем и объектом.

Простые методы коллаборативной фильтрации:

- User2User рекомендации
- Item2Item рекомендации

	1	2	3	4	5	6	7	8	9	10
Петя										
Маша										
Вася										
Катя										

User2User рекомендации

- Введём меру схожести двух пользователей $s(u, v)$, которая тем больше, чем выше сходство между u и v . $s(u, v)$ можно определить как меру Жаккара, скалярное произведение общих рейтингов, корреляцию Пирсона, дисконтированную корреляцию Пирсона. Для пользователя u рассмотрим множество похожих на него пользователей $N(u) = \{v \in U \setminus u \mid s(u, v) > \alpha\}$, где α - пороговое значение (настраиваемый гиперпараметр).
- Оценить рейтинг r_{ui} , который пользователь u поставил бы объекту i , можно, используя рейтинги, которые ставили похожие на u пользователи. Например:

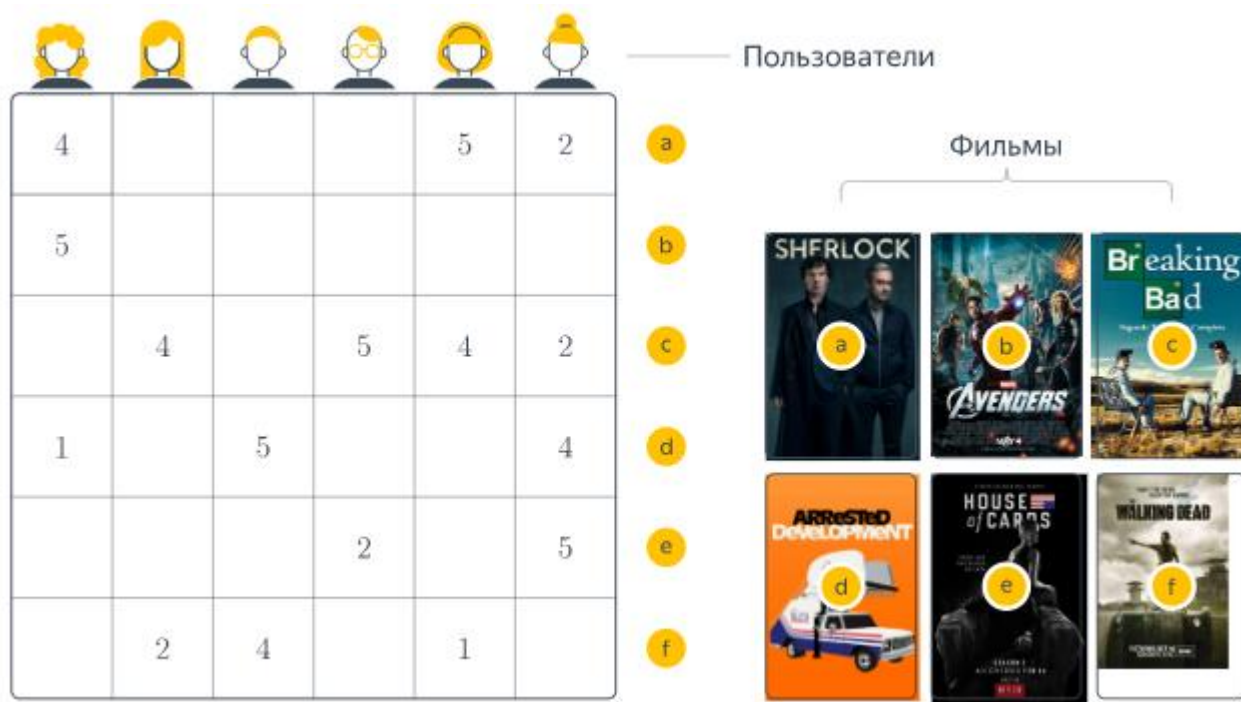
$$\hat{r}_{ui} = \frac{\sum_{v \in N(u)} s(u, v) r_{vi}}{\sum_{v \in N(u)} |s(u, v)|}$$

- Усовершенствование метода оценивания возможно с учетом диапазона оценивания и отклонения от среднего, а также дисперсии оценок пользователя:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N(u)} s(u, v) (r_{vi} - \bar{r}_v)}{\sum_{v \in N(u)} |s(u, v)|}$$

$$\hat{r}_{ui} = \bar{r}_u + \sigma_u \frac{\sum_{v \in N(u)} s(u, v) (r_{vi} - \bar{r}_v) / \sigma_v}{\sum_{v \in N(u)} |s(u, v)|}, \text{ где } \sigma_u = \sqrt{\frac{1}{I_u} \sum_{i \in I_u} (r_{ui} - \bar{r}_u)^2}$$

Матрица оценок user-item. Факторизация



Предположим, что каждый пользователь и объект можно закодировать набором из S скрытых признаков, а оценка i -го объекта u -м пользователем равна скалярному произведению соответствующих векторов скрытых представлений x_u и y_i . Тогда если бы матрица оценок была заполнена полностью, её можно было бы представить в виде произведений двух матриц X и Y , составленных по столбцам из скрытых представлений пользователей и объектов:

$$U = X^T \cdot Y$$

Singular value decomposition (SVD, сингулярное разложение) - метод разложения, при котором исходная матрица разбивается на произведение ортогональных матриц (и диагональной матрицы).

Разложив на компоненты user-item матрицу, можно их вновь перемножить и получить "восстановленную" матрицу, которая будет являться предсказанием оценок в исходной матрице.

Item2Item рекомендации

- Введём меру схожести двух объектов $s(i, j)$. Рейтинг \hat{r}_{ui} , который бы пользователь u поставил объекту i , можно определить аналогично user2user подходу, рассмотрев множество $N(i)$ близких к i объектов:

$$\hat{r}_{ui} = \frac{\sum_{j \in N(i)} s(i, j) r_{uj}}{\sum_{j \in N(i)} |s(i, j)|}$$

- Меру схожести объектов можно задать как взвешенное косинусное расстояние:

$$s(i, j) = \frac{\sum_{u \in U_i \cap U_j} (r_{ui} - \bar{r}_u)(r_{uj} - \bar{r}_u)}{\sqrt{\sum_{u \in U_i \cap U_j} (r_{ui} - \bar{r}_u)^2} \sqrt{\sum_{u \in U_i \cap U_j} (r_{uj} - \bar{r}_u)^2}}$$

где U_i - множество пользователей, оценивших товар i ,

\hat{r}_u – средняя оценка пользователя (не объекта).

Особенности коллаборативной фильтрации

- Они не опираются ни на какую дополнительную информацию кроме матрицы оценок R_{ui} , предполагая, что этого должно быть достаточно для оценки схожести пользователей и товаров;
- Предложенные методы не применимы для новых объектов и пользователей – для них просто нет истории или она недостаточно информативна для того, чтобы методы могли давать более-менее точные оценки;
- Так как методы коллаборативной фильтрации основаны только на истории прошлых взаимодействий, то эти методы не предполагают открытия новых интересов у пользователя, они способны только работать с уже имеющимися.

Content-based рекомендации

- Content-based подход основан на измерении похожести между объектами на основе их содержания. Входом для content-based модели являются разные контентные признаки и характеристики объекта (например, текст статьи, время публикации, картинки), а выходом является некоторое числовое представление объекта (эмбеддинг).
- Оценить рейтинги объектов можно на основе известных контентных эмбеддингов $e_i \in \mathbb{R}^n$ для каждого объекта, определив скалярное произведение (или косинусное расстояние) до оценённых пользователем объектов:

$$\hat{r}_{ui} = \max_{j \in I_u, r_{uj} > \alpha} \rho(e_i, e_j) r_{uj},$$

где ρ – скалярное произведение или косинусное расстояние между двумя векторами,

I_u – множество оценённых пользователем объектов,

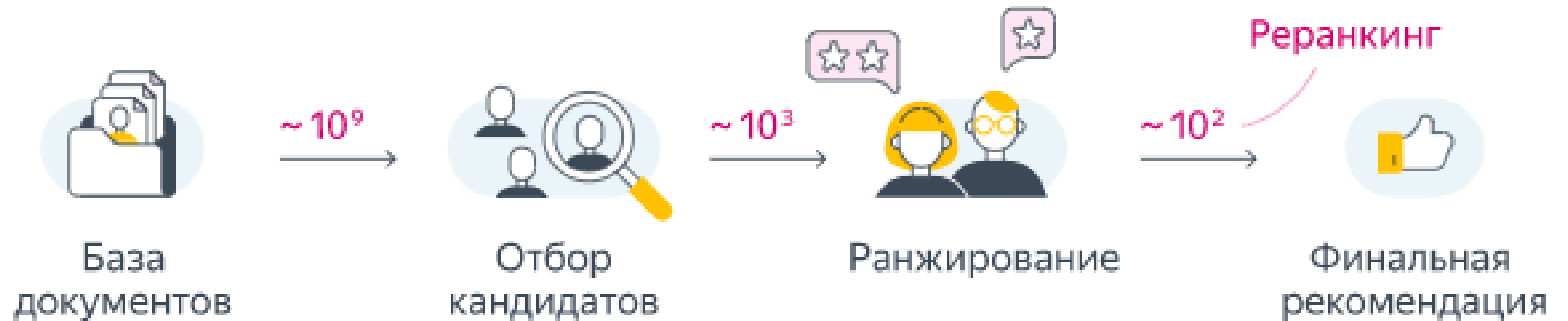
α – гиперпараметр.

- Таким образом, высокие рейтинги получают объекты, похожие на те, что понравились пользователю.

Особенности контентного подхода

- Плюс контентного подхода: в отличие от чисто коллаборативного подхода, он одинаково хорошо работает на новых и старых объектах, так как контентные модели основаны только на статичной контентной информации, которая всегда доступна.
- Минус контентного подхода: отсутствие рекомендации объектов, которые следуют напрямую из действий пользователей.
- Существуют гибридные модели, совмещающие в себе коллаборативный и контентный сигналы, например, DSSM.

Классический пайплайн рекомендательной системы



Метрики работы рекомендательных систем

- Полнота (Coverage) - доля рекомендованных объектов $I_{recommended}$ среди всех объектов I :

$$Coverage = \frac{|I_{recommended}|}{|I|}$$

- Новизна (Novelty) – мера информации, определяющая новизну объекта для пользователя среди списка его рекомендаций R :

$$Novelty_{user} = \frac{1}{|R|} \sum_{i \in R} -\log(P(i))$$

где P_i – вероятность рекомендации i -о объекта случайному пользователю.

- Разнообразие (Diversity) - способность модели рекомендовать разные по содержанию объекты.

$$ILS_{user} = \frac{1}{R} \sum_{i \in R} \sum_{j \in R} \text{sim}(i, j)$$

где sim – попарная схожесть между объектами, R – набор рекомендованных пользователю объектов.

- Serendipity - способность рекомендовать такие объекты, которые не только релевантны для пользователя, но ещё и существенно отличаются от того, с какими объектами пользователь взаимодействовал в прошлом.

Пример реализации рекомендательной системы

```
import numpy as np
import pandas as pd
df = pd.read_csv('data.csv')
n_users = df['user'].unique().shape[0] #Проверяем уникальность пользователей и объектов
n_items = df['item'].unique().shape[0]
df['item'] = df['item'].apply(lambda x: np.where(df['item'].unique() == x)[0][0] + 1) #Приводим значения item к диапазону от 1 до числа объектов
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(df, test_size=0.20)
train_data_matrix = np.zeros((n_users, n_items)) # Создаём две user-item матрицы
for line in train_data.itertuples(): train_data_matrix[line[1] - 1, line[2] - 1] = line[3]
test_data_matrix = np.zeros((n_users, n_items))
for line in test_data.itertuples(): test_data_matrix[line[1] - 1, line[2] - 1] = line[3]
```

Пример реализации рекомендательной системы

```
from sklearn.metrics import mean_squared_error
def rmse(prediction, ground_truth): #Задаем метрику работы
    prediction = prediction[ground_truth.nonzero()].flatten()
    ground_truth = ground_truth[ground_truth.nonzero()].flatten()
    return mean_squared_error(prediction, ground_truth)**0.5
import scipy.sparse as sp
from scipy.sparse.linalg import svds
u, s, vt = svds(train_data_matrix, k=10) # SVD
s_diag_matrix = np.diag(s)
X_pred = np.dot(np.dot(u, s_diag_matrix), vt)
print('User-based CF MSE: ' + str(rmse(X_pred, test_data_matrix)))
```

Ассоциативные правила

- Обучение на ассоциативных правилах (Associations rules learning — ARL) - метод поиска взаимосвязей (ассоциаций) в датасетах (itemsets).
- В общем виде ARL можно описать как «Кто купил x , также купил y ». При помощи ARL алгоритмов находятся «правила» совпадения items внутри одной транзакции, которые потом сортируются по их силе.
- Пусть имеется некий датасет (или коллекция) D : $D = d_0...d_j$, где d — уникальная транзакция-itemset (например, кассовый чек). Внутри каждой d представлен набор items (i-item), например в бинарном виде. Принято каждый itemset описывать через количество ненулевых значений (k-itemset).

Метрики ассоциативных правил

- Support (поддержка) - показатель частотности itemset во всех анализируемых транзакциях:

$$supp(X) = \frac{|\{t \in T; X \in t\}|}{|T|}$$

где X — itemset, содержащий в себе i -items, а T — множество транзакций.

Например, для датасета, содержащего x_1 и x_2 :

$$supp(x_1 \cup x_2) = \frac{\sigma(x_1 \cup x_2)}{|T|}$$

где $\sigma(x_1 \cup x_2)$ - количество транзакций, содержащих x_1 и x_2 .

- Confidence (достоверность) – показатель частоты срабатывания взятого правила:

$$conf(x_1 \cup x_2) = \frac{supp(x_1 \cup x_2)}{supp(x_1)}$$

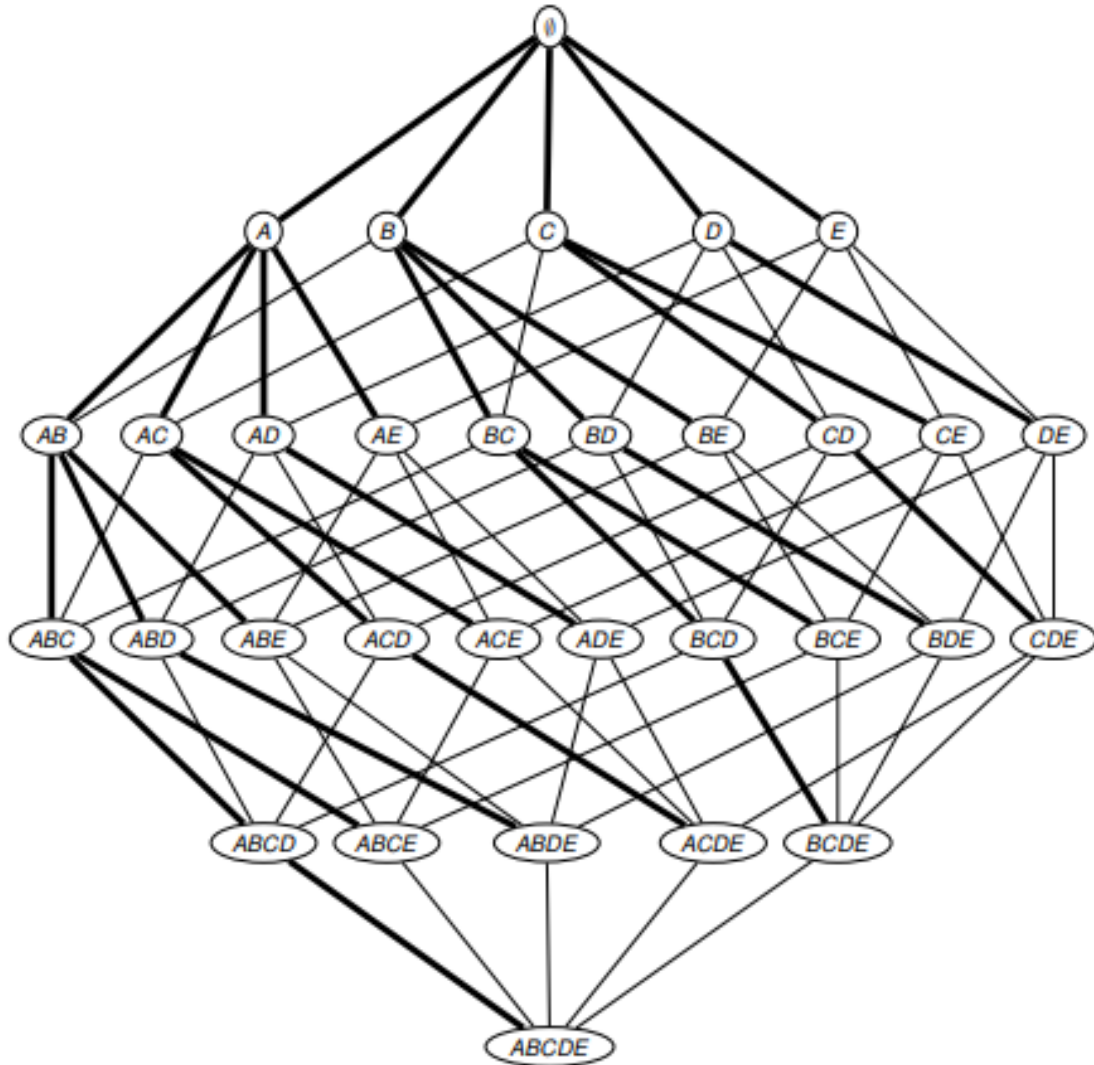
- Lift (зависимость) – показатель зависимости items друг от друга:

$$lift(x_1 \cup x_2) = \frac{supp(x_1 \cup x_2)}{supp(x_1) \times supp(x_2)}$$

- Conviction (убедительность) - «частотность ошибок» взятого правила:

$$conv(x_1 \cup x_2) = \frac{1 - supp(x_2)}{1 - conf(x_1 \cup x_2)}$$

Алгоритм Apriori



Apriori использует следующее утверждение: если $X \subseteq Y$, то $\text{supp}(X) \geq \text{supp}(Y)$. Отсюда следуют 2 свойства:

- если Y встречается часто, то любое подмножество, содержащее X , так же встречается часто;
- если X встречается редко, то любое супермножество Y , содержащее X , так же встречается редко.

Apriori алгоритм по-уровнево проходит по префиксному дереву и рассчитывает частоту встречаемости подмножеств X в D . Таким образом, в соответствии с алгоритмом:

- исключаются редкие подмножества и все их супермножества;
- рассчитывается $\text{supp}(X)$ для каждого подходящего кандидата X размера k на уровне k .

Пример поиска ассоциативных правил

```
import pandas as pd
dataset = pd.read_csv('data.csv', header = None)
dataset.fillna(method = 'ffill',axis = 1, inplace = True)
transactions = [] # создаим матрицу транзакций
for i in range(0, n_transaction):
    transactions.append([str(dataset.values[i,j]) for j in range(0, 20)])

import apyori
from apyori import apriori
rules = apriori(transactions, min_support = 0.00030, min_confidence = 0.05, min_lift = 3,
max_length = 2, target = "rules")
association_results = list(rules)
print(association_results[0])
```

Пример поиска ассоциативных правил

```
for item in association_results:
    pair = item[0]
    items = [x for x in pair]
    print("Rule : ", items[0], " -> " + items[1])
    print("Support : ", str(item[1]))
    print("Confidence : ",str(item[2][0][2]))
    print("Lift : ", str(item[2][0][3]))
    print("=====")
```

Пример поиска ассоциативных правил

```
import shutil, os

try:
    from StringIO import StringIO
except ImportError:
    from io import StringIO

import json #преобразуем в json
output = []

for RelationRecord in association_results:
    o = StringIO()
    apyori.dump_as_json(RelationRecord, o)
    output.append(json.loads(o.getvalue()))
```

Пример поиска ассоциативных правил

```
data_df = pd.DataFrame(output)
pd.set_option('display.max_colwidth', 1)

from IPython.display import display, HTML

display(HTML(data_df.to_html()))
```