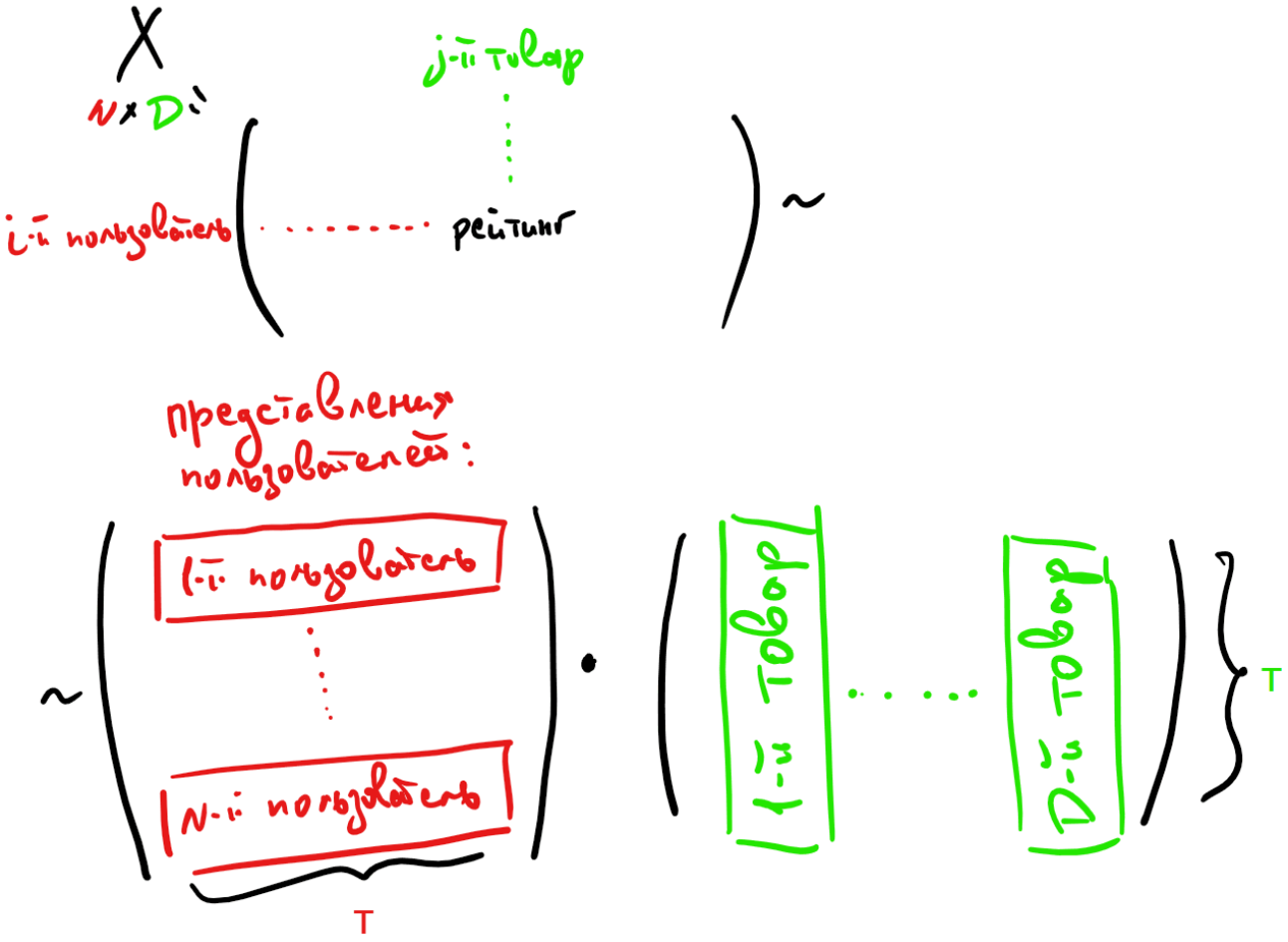
9.2. Рекомендации на основе матричных разложений

Введение

Тут про рекомендации на основе матрицы оценок user-item. Строки - товары, столбцы - юзеры. В каждую клетку записываем либо оценку, либо пропуск. Задача в том, чтобы заполнить пропуски.

Связь с задачей матричной факторизации

Каждого юзера и товара можно закодировать набором из S скрытых признаков, а оценка i -го товара u-м юзером равна скалярному произведению соответствующих векторов скрытых представлений x_u и y_i. Тогда, если бы матрица оценок была заполнена полностью, то её можно было бы представить в виде произведения матриц X и Y, составленных по столбцам из скрытых представлений товаров и юзеров. $U = X^T \cdot Y$



В таком случае, мы бы могли рекомендовать каждому юзеру товары с наибольшим значением оценки из соответствующей строки, но в реальности матрица оценок сильно разрежена. То есть задача в том, чтобы по имеющемуся набору оценок восстановить латентные векторы для юзеров и товаров, а затем уже заполнить пропуски в матрице оценок. Сделать это можно с помощью SVD и стохастического градиентного спуска. В случае SVD наши значения получатся слишком шумными из-за того, что матрица сильно разрежена, а также нам придётся пересчитывать значения при добавлении новых юзеров или товаров. Градиентный спуск тоже не идеален. Гораздо лучше работает **Alternating Least Squares (ALS)**. Далее речь про явные оценки.

Постановка задачи

Пусть матрицы X и Y размерами S x N и S x D, где N - число юзеров, D - число товаров. Пусть множетсво R - множество пар юзеров и товаров, для которых оценка известна.
$\hat{r}_{ui} = x_u^T y_i$

Мы хотим научиться как можно лучше приближать известные рейтинги:

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

С регуляризацией функция потерь выглядит так:

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

Alternating Least Squares (ALS)

Сложность в том, что нам нужно оптимизировать сразу два набора объектов (матрицы X и Y).
Суть алгоритма в том, что мы фиксируем, например, матрица Y, затем вычисляется оптимизируется функция потерь от X путём решения задачи наименьших квадратов, затем наоборот, фиксируем X и считаем по Y, до тех пор, пока не достигнем нужно сходимости. Момент в том, что каждая строка вычисляемой матрицы вычисляется независимо от других строк, то есть мы можем использовать параллельные вычисления.

Implicit ALS (IALS)

Теперь про неявные оценки. Так как за неявную оценку можно принять факт взаимодействия, то мы можем полностью заполнить матрицу оценок. Поставим в клетки 1, если юзер положительно взаимодейтсвовал в товаром, иначе, если негативно или вообще не взаимодействовал, то 0 (preference). Однако такой подход слишком прост, так как степерь взаимодействия (confidence) может сильно различаться.

$$c_{ui} = 1 + \alpha |r_{ui}| \text{ (степень уверенности в } p_{ui})$$

где a - некоторая константа.
Тогда хороша следующая функция потерь:

$$\sum_{\forall u,i} c_{ui} (p_{ui} - x_u^T y_i)^2 + \lambda \sum_{\forall u} ||x_u||^2 C_u + \lambda \sum_{\forall i} ||y_i||^2 C_i$$

IALS: оптимизация

Немного поигравшись с предыдущей функцией потерь:

$$\Big(Y^T Y + \lambda C_u I + \sum_{\forall i: p_{ui} \neq 0} (c_{ui} - 1) y_i y_i^T\Big)^{-1} \Big(\sum_{\forall i: p_{ui} \neq 0} c_{ui} p_{ui} y_i\Big)$$

Заметим, что Y^TY не зависит от u, поэтому мы можем посчитать его один раз для всех x_u

Заметим, что и в ALS и в IALS константой как факта взаимодействия, так и стандартную уверенность необязательно ставить 1. Например, события «пользователь не посмотрел популярный фильм» и «пользователь не посмотрел редкий фильм» могут иметь для нас разный вес.

И в обоих алгоритмах $r_{ui} \approx x_u y_i + b_u + b_i + \mu$, где играют b_i, b_u роль некоторых априорных усреднённых оценок пользователя и объекта соответственно, а μ является глобальной априорной константой

FunkSVD

Тот же ALS, только оптимизация проводится с помощью стахостического градиентного спуска. Используется редко, так как вычисления сложно распараллелить.

Singular Value Decomposition with implicit feedback (SVD++)

Комбинирует использование явных и неявных оценок, используя историю неявных оценок товаров пользователем.

$$r_{ui} \approx \left(x_u + \frac{1}{\sqrt{|\{j|p_{uj} \neq 0\}|}} \sum_{\forall j: p_{uj} \neq 0} \hat{y}_j\right)^T y_i + b_u + b_i + \mu$$

Важно отметить, что вектора $\hat{y}_j$ не совпадают с векторами y_i. Это своего рода «неявные» вектора айтемов.

Collaborative Filtering with Temporal Dynamics (timeSVD++)

Тот же SVD++, только учитывает порядок оценок айтемов в истории.

$$r_{ui}(t) \approx \left(x_u(t) + \frac{1}{\sqrt{|\{j|p_{uj} \neq 0\}|}} \sum_{\forall j: p_{uj} \neq 0} \hat{y}_j\right) y_i + b_u(t) + b_i(t) + \mu$$

SLIM (Sparse Linear Methods)

Вышеописанные методы показывают хороший результат, но требуют много усилий для работы в онлайн сервисах. Возникает потребность в лёгких моделях, которые не сильно

хуже по качеству.

Пусть A - бинарная матрица N x D user-item взаимодействий, результатом алгоритма будет взвешеванием событий из истории юзера.

$$\hat{a}_{ui} = \sum_j w_{ij} a_{uj}$$

При этом $w_{ij} \geq 0$. А w_{ii} = 0, чтобы однажды не получить единичную матрицу. В результате вес является мерой схожести. Нужно только научиться оптимизировать веса.
Используем MSE с L1, L2 регуляризаторами:

$$\frac{1}{2} \sum_{u,i} (a_{ui} - \sum_j w_{ij} a_{uj})^2 + \lambda \sum_{i,j} |w_{ij}| + \frac{\beta}{2} \sum_{i,j} (w_{ij})^2 \to \min_W$$

Данная задача можно решить координатным спуском: примерно как ALS.
Алгоритм модели:

1. Рассчитываем вектор взаимодействия пользователя $(u_{ui})_{i=1}^D$
2. Считаем ata_{ui} для всех непросмотренных объектов
3. Отбираем топ k непросмотренных объектов по $\hat{a}_{ui}$

В итоге, из-за L1 регуляризации матрица W получается разреженной, что хорошо.

Итоги

P.S. можно построить индекс для функций близости, чтобы быстро, но приближённо считать.

Итого, в таком случае принцип работы:

1. обучаются эмбединги товаров и юзеров
2. для представлений эмбеддингов строится индекс
3. в рантайме по вектору юзера происходит приближённый поиск n самых релевантных объектов, таким образом генерируется список кандидатов в рекомендации
4. список кандидатов обрабатывается последующими моделями машинного обучения

Главный минус методов, основанных на матричной факторизации, в том, что они используют информацию о взаимодействии юзеров и товаров, но не используют информацию о них самих.