2 Этап
Для начала формализуем задачу:

Введем ряд обозначений. Пусть у нас есть множество пользователей U и множество объектов I. Для каждого пользователя $u \in U$ есть множество объектов $I_u \subset I$, с которыми он взаимодействовал и которым поставил рейтинги $R_u = \left(r_{ui}\right)_{i \in I_u}$.

Рейтинг (его также называют фидбеком) – это некоторая характеристика взаимодействия пользователя с объектом; про него можно думать, как про некоторый таргет, который мы выбрали для оптимизации рекомендательной системы. Таким образом, задачу рекомендательных систем можно переформулировать в следующем виде: для каждого пользователя $u \in U$ необходимо оценить значение r_{ui} для $i \in I \setminus I_u$ и выбрать несколько товаров с наибольшим $\widehat{r_{ui}}$. Иными словами, надо научиться среди непоказанных пользователю товаров находить те, которые заинтересовали бы его больше всего.

В нашем случае фидбеком является сам факт покупки товара.

Имеющиеся решения их плюсы и минусы

Виды рекомендательных систем
- **Кластеризация пользователей**
- **Summary-based** (неперсональные)
- **Content-based** (основанные на описании товара)
- **Коллаборативная фильтрация**
- **Матричная факторизация**

Explicit / implicit подходы:

Явный (Explicit) подход: В этом подходе пользователи явно оценивают товары или элементы, предоставляя оценки или отзывы.

Пример явных таргетов: Рейтинги, лайк, дизлайк.

Неявный (Implicit) подход: В этом подходе информация о предпочтениях пользователей извлекается из неявных сигналов, таких просмотр карточки, добавление в корзину или избранное, поисковый запрос, заказ.

В нашем случае по сути нам известны только неявные таргеты.

Кластеризация пользователей

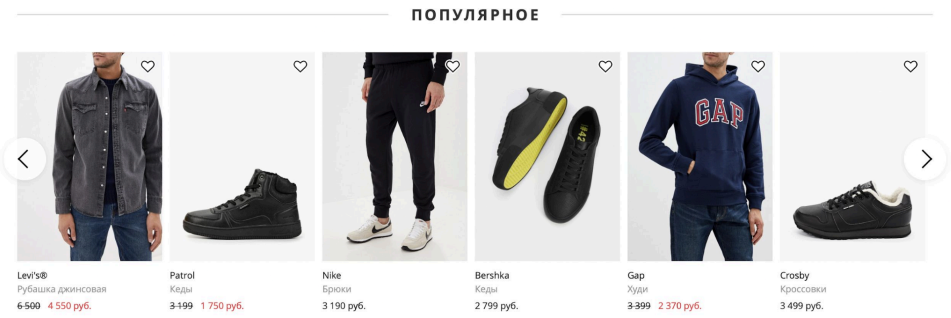
• Выберем меру схожести пользователей **sim(u, v)** по истории их оценок
• Объединим пользователей в кластеры так, чтобы похожие пользователи попали в один кластер
• Оценку пользователя объекту будем предсказывать среднюю оценку всего кластера по этому объекту
Кластеризация пользователей

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

Проблемы:
● Нечего рекомендовать **новым/нетипичным пользователям**
● 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

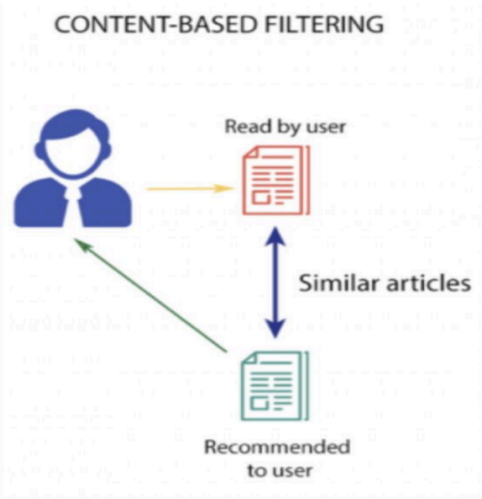
Summary-based рекомендации

Подход неперсонализированных рекомендаций — это расчет рекомендаций на основании глобального рейтинга. В качестве рейтинга может использоваться средняя оценка товара, отношение лайков к просмотрам, количество скачиваний, или другая метрика, специфичная для системы. В нашем же случае, это могут быть наиболее покупаемые товары.
Допускают таргетинг по региону или времени.
Проблема — **игнорирование личных предпочтений** пользователя.
Также большая проблема в том, что мы не будем рекомендовать пользователю новые товары, потому что для них попасть в топ-рейтинга быстро очень тяжело.



Content-based рекомендации

Основная идея - использовать характеристики объекта для поиска похожих объектов.
Функция оценки - обычно скалярное произведение или косинусное расстояние.
Процесс построения рекомендаций - ищем наиболее похожие на те объекты, с которыми пользователь уже взаимодействовал.



Если есть хорошие признаковые описания пользователей и объектов (и только они), тогда

$$u \sim f_u$$
$$i \sim f_i$$

Можно решать как обычную задачу обучения с учителем

$$\{([f_u, f_i], r_{ui})\}$$

Цель: $u \to i_1, \ldots, i_k : \hat{r}_{ui_1} \geq \hat{r}_{ui_2} \geq \ldots$

Есть плюс в отсутствии проблемы холодного старта.
Минус в отсутствии контекста и учета опыта других пользователей.
Но в нашем случае, к сожалению, content-based рекомендации не подойдут, т.к. информации о самих товарах у нас нет.

Какие бывают лоссы в задаче с учителем в recsys:

- **MSE/MAE**
  Используется для регрессионных задач, где предсказания модели сравниваются с фактическими значениями. Функция ошибки выражается как среднее значение квадрата разности (абсолютное значение разности) между предсказаниями и истинными значениями.
- **Log Loss**
  Часто используется в задачах классификации и рекомендациях вероятности. Эта функция штрафует модель за недостоверные предсказания вероятностей.
- **Ranking Loss**
  Эта функция ошибки оценивает качество модели по ее способности правильно упорядочивать предложения, товары или рекомендации по отношению к остальным.
- **Cross-Entropy Loss**
  В рекомендательных системах Cross-Entropy Loss часто применяется в контексте задачи классификации или

предсказания вероятностей, когда рекомендуемые элементы могут быть рассмотрены как классы.

Коллаборативная фильтрация

Это метод рекомендации, при котором анализируется только реакция пользователей на объекты: оценки, которые выставляют пользователи объектам.

- **Memory-based**
- **Model-based**

Memory-based collaborative filtering

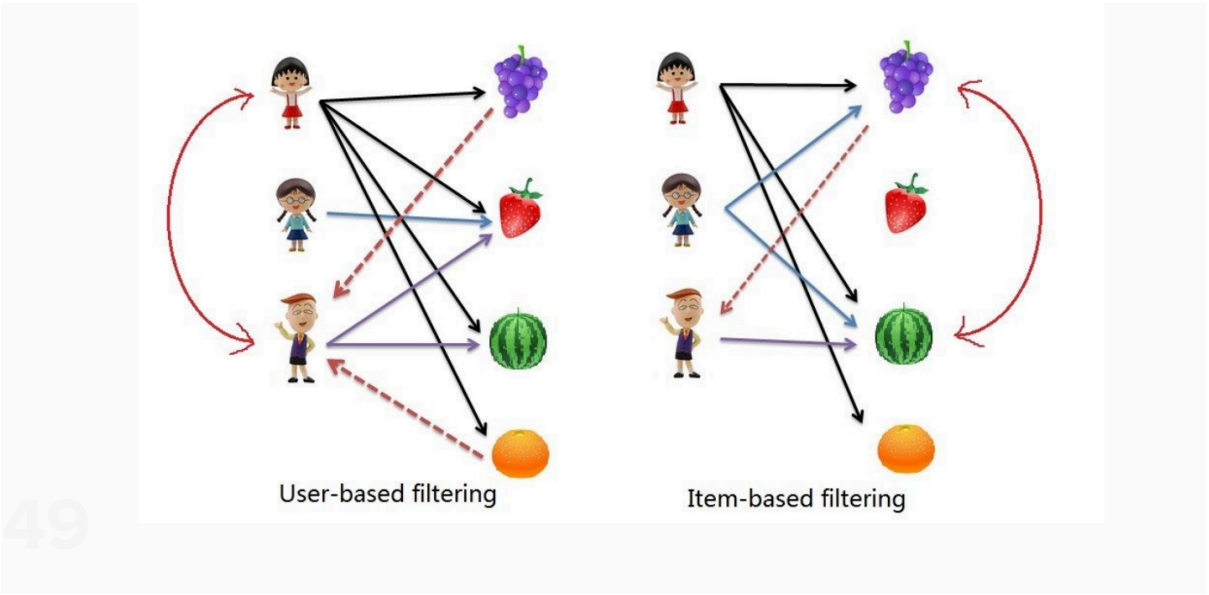
Эвристический метод основанный на соседстве, заключается в составлении разреженной матрицы оценок, ее «запоминании» и поиске по метрике близости ближайших товаров или пользователей. Можно искать по метрике близости cosine similarity или др

Два подхода:

- Item-based - сравнивает элементы и рекомендует похожие на те, что пользователь ранее выбирал.
- User-based - сравнивает предпочтения пользователей и рекомендует элементы, оцененные похожими пользователями.

Как использовать:

- Найти похожие объекты на то, с чем пользователь взаимодействовал
- Рекомендовать объекты из тех, с которыми взаимодействовали похожие пользователи



По пользователям (User-based)

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_v \text{sim}(u,v)(r_{vi} - \bar{r}_v)}{\sum_v \text{sim}(u,v)}$$

По товарам (Item-based)

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum_j \text{sim}(i,j)(r_{uj} - \bar{r}_j)}{\sum_j \text{sim}(i,j)}$$

Что лучше? **item-based** или **user-based**?

- Когда пользователей много (почти всегда), задача поиска ближайшего соседа становится плохо вычислимой
- Оценка близости товаров гораздо более точная, чем оценка близости пользователей
- В user-based варианте описания пользователей, как правило, сильно разрежены (товаров много, оценок мало). Проблема: сколько соседей не бери, список товаров, которые в итоге можно порекомендовать, получается очень небольшим
- Нечего рекомендовать новым / нетипичным пользователям. Для таких пользователей мы все еще не можем найти похожих.

Особенности коллаборативной фильтрации

Выделим ключевые особенности методов, основанных на коллаборативной фильтрации, о которых следует помнить при разработке рекомендательных систем:

- Они не опираются ни на какую дополнительную информацию кроме матрицы оценок R_{ui}, предполагая, что этого должно быть достаточно для улавливания качественного сигнала о 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, рекомендательная система, построенная исключительно на их основе будет постепенно вгонять пользователя в информационный пузырь: эти методы не предполагают открытия новых интересов у пользователя, они способны только эксплуатировать уже имеющиеся
- Ресурсоемкость вычислений. Для того, чтобы делать предсказания нам нужно держать в памяти все оценки всех пользователей. (что являются большим минусом, т.к. матрица у нас большая)

.

Model-based

Метод основанный на модели, заключается в общении модели, которая аппроксимирует близость товаров и пользователей. Предполагает, что сходство между пользователями и объектами вызвано некоторой скрытой низкоразмерной структурой данных.

Получить векторные представления пользователей и объектов на основе данных об их взаимодействиях.

Функция оценки релевантности для пар (пользователь, объект)

$$f_{ui} : Users \times Items \rightarrow Relevance$$

Вектора получаем за счет фиксирования какой-либо функции оценки и решения задачи оптимизации

$$L\left(R, \widehat{R}\right) \rightarrow min$$

SVD

SVD (Singular Value Decomposition) - это метод разложения матрицы на три более простые матрицы, чтобы найти скрытые факторы, описывающие взаимодействие между пользователями и товарами в рекомендательной системе.

В теореме о сингулярном разложении утверждается, что у любой матрицы **A** размера $n \times m$ существует разложение в произведение трех матриц: U, Σ и V:

$$\underset{n\times m}{\boldsymbol{A}} = \underset{n\times n}{\boldsymbol{U}} \times \underset{n\times m}{\boldsymbol{\Sigma}} \times \underset{m\times m}{\boldsymbol{V}^T}$$

$$\boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{I}_n, \quad \boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}_m,$$

$$\boldsymbol{\Sigma} = \mathrm{diag}\left(\lambda_1, \ldots, \lambda_{\min(n,m)}\right), \quad \lambda_1 \geqslant \ldots \geqslant \lambda_{\min(n,m)} \geqslant 0$$

Усеченное SVD (Singular Value Decomposition)

$$\lambda_{d+1}, \ldots, \lambda_{\min(n,m)} := 0$$

$$\underset{n\times m}{\boldsymbol{A}'} = \underset{n\times d}{\boldsymbol{U}'} \times \underset{d\times d}{\boldsymbol{\Sigma}'} \times \underset{d\times m}{\boldsymbol{V}'^{T}}$$

Полученная матрица $\mathbf{A}'$ хорошо приближает исходную матрицу $\mathbf{A}$ и, более того, является **наилучшим** низкоранговым приближением с точки зрения средне-квадратичного отклонения.

У нас была матрица, мы разложили ее в произведение трех матриц. При чем разложили не точно, а приблизительно.

С помощью градиентного спуска или других методов оптимизации находятся обновленные значения матриц U и V, которые минимизируют функцию потерь.

Особенности:

- Не так хорошо для предсказания значений (в качестве регрессии)
- Подходит для построения топ-N рекомендаций
    - Для каждого пользователя считаем скалярное произведение со всеми объектами
    - Выбираем топ объектов по полученным значениям

Основной минус в том, что SVD может столкнуться с проблемами масштабирования на очень больших объемах данных, таких как наш набор данных.

Alternating Least Squares (ALS)

Постановка задачи

Пусть x_u, y_i - скрытые представления пользователей и объектов соответственно размерности T. Запишем эти векторы по строкам в матрицы X и Y размера $S \times N$ и $S \times D$ соответственно, где N - количество пользователей, а D - количество объектов.

Обозначим через R множество таких пар (u, i) пользователей и объектов, для которых имеются явно проставленные оценки.

Предсказать рейтинги мы будем как скалярное произведение скрытых представлений

$$\widehat{r_{ui}} = x_u^T y_i$$

В результате мы приходим к следующей задаче оптимизации. Мы хотим научиться как можно лучше приближать известные рейтинги:

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

Добавив регуляризацию получаем следующую функция потерь:

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

Если мысленно заморозить параметры, соответствующие латентным факторам пользователей, задача оптимизации латентных представлений объектов сводится к задаче наименьших квадратов.

В цикле до сходимости:

- Фиксируем матрицу X (скрытые представления пользователей);
- Решаем задачу L2-регуляризованной регрессии для каждого товара и находим оптимальную матрицу Y
- Фиксируем матрицу Y (скрытые представления объектов);

- Решаем задачу L2-регуляризованной регрессии для каждого пользователя и находим оптимальную матрицу X

ALS лучше масштабируется и может быть распараллелен, что является большим плюсом.

Раньше мы работали с матрицей R как с матрицей рейтингов, явно проставленных пользователем. Но, например в нашем случае, у нас только неявный фидбек.

Неявным фидбеком является в том числе и факт взаимодействия, поэтому мы можем заполнить всю матрицу user-item целиком: на тех позициях, где пользователь положительно взаимодействовал с объектом, поставим 1, а на тех, где взаимодействие было негативным или его вообще не произошло, поставим 0.

$$p_{ui} = \{1,\ r_{ui} > 0,\ 0,\ r_{ui} \leq 0 \text{ или } r_{ui} \text{ не определено}$$

Введём ещё степень уверенности (confidence), отражающую уверенность в оценке пользователя:

$c_{ui} = 1 + a\left|r_{ui}\right|$, где a – некоторая константа.

Получим следующую функцию потерь:

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

Она позволяет:

- Учитывать неявный фидбек, которого обычно на порядок больше, чем явного,

- Регулировать степень уверенности в действиях пользователей.

1. Масштабируемость: ALS позволяет эффективно работать с большим объемом данных и может быть распараллелен для обучения на множестве данных.

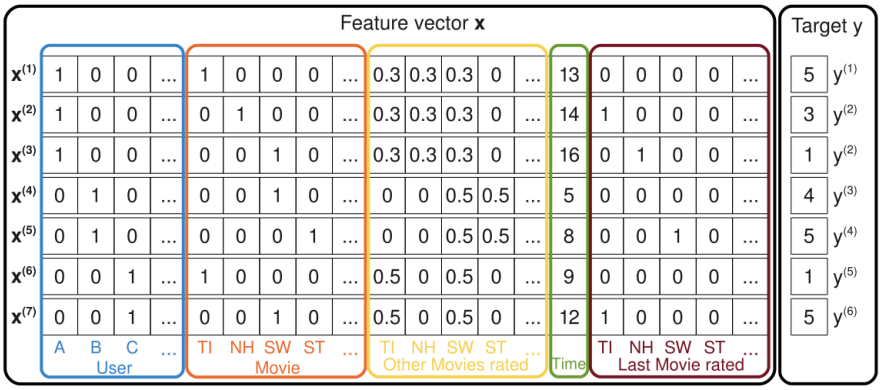
2. Учет взаимодействия: ALS умеет учитывать интеракции между пользователями и товарами без необходимости других данных, что подходит для задачи рекомендаций.

Главная его проблема — он оптимизирует не ту функцию потерь. Вспомним формулировку задачи построения рекомендательной системы. Мы хотим получить отношение порядка на множестве, а вместо этого оптимизируем среднеквадратическое отклонение.

Factorization Machines

https://web.archive.org/web/20230330055726/https://www.csie.ntu.edu.tw/~b97053/paper/Rendle2010FM.pdf

Допустим что целевая переменная зависит не только от самих признаков, но и еще от их попарного взаимодействия (полиномиальная регрессия 2-ого порядка). Оптимизация параметров происходит с помощью стохастического градиентного спуска (SGD). Первая часть вектора кодирует пользователя, вторая — товар. После можно добавить дополнительные признаки — историю просмотров пользователя (третья часть). Другие дополнительные признаки зависят лишь от данных.



Bayesian Personalized Ranking

https://arxiv.org/ftp/arxiv/papers/1205/1205.2618.pdf

Известно, с какими айтемами взаимодействовал пользователь. Будем считать, что это положительные примеры, которые ему понравились. Остается множество айтемов, с которым пользователь не взаимодействовал. Мы не знаем, какие из них пользователю будут интересны, а какие нет, но мы наверняка знаем, что далеко не все из этих примеров окажутся положительными. Сделаем грубое обобщение и будем считать отсутствие взаимодействия отрицательным примером.

Будем сэмплировать тройки {пользователь, положительный айтем, отрицательный айтем} и штрафовать модель, если отрицательный пример оценен выше положительного.

Weighted Approximate-Rank Pairwise

Добавим к предыдущей идее adaptive learning rate. Будем оценивать обученность системы, исходя из количества семплов, которые нам пришлось просмотреть, чтобы для данной пары {пользователь, положительный пример} найти отрицательный пример, который система оценила выше положительного.

Если мы нашли такой пример с первого раза, значит штраф должен быть большой. Если пришлось долго искать, значит система уже неплохо работает и штрафовать так сильно не нужно.

Word2Vec

Есть и другие подходы к получению эмбеддингов из разреженной матрицы взаимодействий пользователей с объектами. Один из них заимствован из языковых NLP-моделей и носит название Word2Vec.

Наши предложения – последовательность товаров, купленные пользователем за сессию (30 минут, 2 часа, неделя, 2 недели, месяц)

Слова – товар

Рекомендации – самые ближайшие товары к рассматриваемому по Word2Vec

Это I2I рекомендации

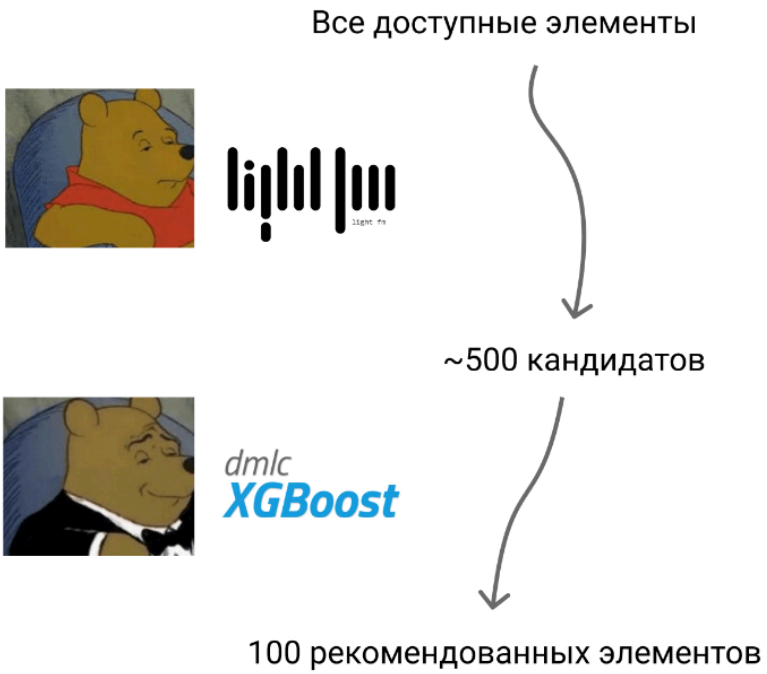
Плюсы:

- Word2Vec способен улавливать семантические отношения между товарами в последовательности покупок.
- Модель Word2Vec может помочь выявить схожие товары и предложить их в качестве рекомендаций.
- Временные интервалы могут использоваться в качестве контекста для обучения Word2Vec, что позволяет учитывать динамику предпочтений пользователей.

Минусы:

- Word2Vec может не учитывать индивидуальные предпочтения пользователей, так как модель обучается на данных о всех пользователях.
- Модель Word2Vec склонна к захвату краткосрочных зависимостей, что может недооценить долгосрочные предпочтения пользователей.
- Word2Vec может испытывать затруднения с рекомендациями для новых товаров или таких, которых нет в обучающем наборе данных.

Двухуровневая модель рекомендаций.



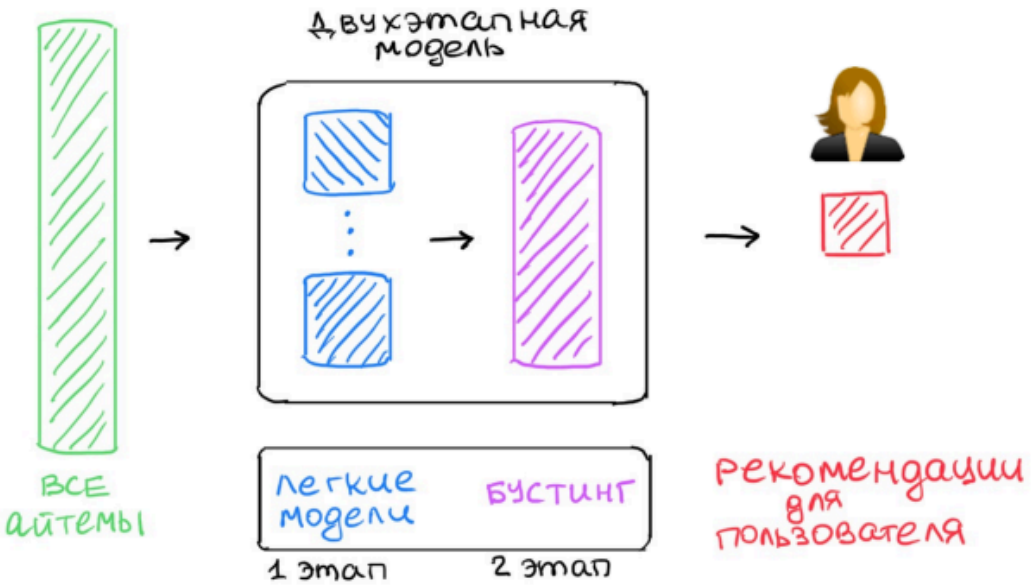
Допустим мы обучили простые модели (SVD, ALS и т.д.). Что делать дальше?

В чем недостатки этих моделей:

1) Они не учитываю временую составляющую. Вкусы с течением времени у пользователя меняются, поэтому хочется придать вес недавним взаимодействиям и снизить старым. В простых моделях при построении матрицы user-item это не учитывается в больших случаях.
2) Простые модели линейные, нам может хотеться построить более сложную нелинейную модель.
3) Зачем нам использовать одну из моделей, если мы можем заблендить результаты нескольких моделей.

Логично воспользоваться двухэтапной моделью, которая будет ранжировать результаты простых моделей.

Этап предикта:



Отбор кандидатов важен, т.к. без него через модель 2 этого этапа пройдет огромное количество объектов, а вычислительные ресурсы ограничены.

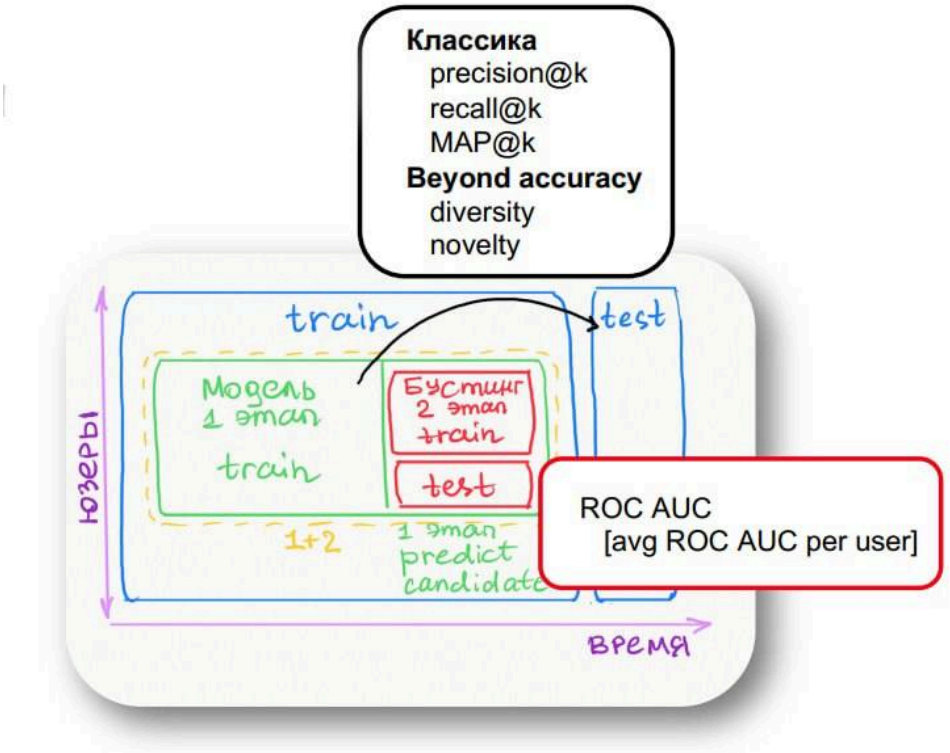
Модель 2-го этапа

Цель: переранжировать кандидатов 1-го этапа таким образом, чтобы метрики выросли.

Особенности подхода заключаются в:

1. Правильная валидация важна
2. Правильный сбор трейна, эксперементы
3. Проблема с ресурсами:
    3.1. Если долго обучается модель 2 этапа
        3.1.1. переобучаем раз в неделю, а применяем каждый день
        3.1.2. переобучаем каждый день только модель для кандидатов 1 этапа
    3.2. Долгий этап применения
        3.2.1. Батчевый подход
        3.2.2. Spark ML

Схема валидация + метрики.



Best practice в электронной коммерции

Как я понимаю, лучшей практикой является двухуровневая модель.

Например, мы отбираем top-N (400) кандидатов с помощью быстрой модели ALS, далее переранжируем их сложной моделью, например LightGBM, и выберем top-k (16).

Выбор

Воспользуюсь 2-ух уровневной модель. Где на 1-ом этапе использую какой-нибудь метод коллаборативной фильтрации (например IALS), а на 2-ом переранжирую данные полученные моделью/моделями на 1-ом этапе.

Также попробую метод связанный с word2vec.