Рекомендательные системы

Дисклеймер: данная лекция предполагает знакомство с рекомендательными системами на базовом уровне в рамках курсов по машинному обучению. Желательно примерно помнить идею колаборативной фильтрации и иметь представление о задаче ранжирования.

Постановка задачи

В задаче рекомендаций рассматривается два типа объектов: пользователи и предметы (users and items). Цель рекомендательной системы — сделать так, чтобы пользователь провзаимодействовал с предметом. В зависимости от конкретной задачи под предметом и взаимодействием может подразумеваться разное. Примеры предметов

- книги в книжном магазине
- фильмы в кинотеатре
- видео на видео-хостинге
- посты и паблики в социальной сети
- вклады в банке

Примеры взаимодействия: посмотреть, оставить комментарий, лайкнуть, подписаться, ну и конечно самое замечательное — купить.

В самом благоприятном случае рекомендательная система располагает данными о следующем:

- о пользователе (пол, возраст, место жительства, история взаимодействий с предметами)
- о предмете (размер, дата выхода, вкус, история взаимодействий пользователей)
- о взаимодействии (тип, дата)

Строго постановка задачи выглядит следующим образом: имея вышеперечисленные данные, для пользователя u выдать упорядоченный набор $i_1, \dots, i_k$, с предметами из которого он точно провзаимодействует.

Общий подход к решению задачи рекомендации состоит из двух этапов:

- отбор кандидатов
- ранжирование кандидатов и формирование выдачи

Первый этап состоит в поиске предметов в огромном каталоге всех имеющихся предметов. Это непростая задача, которую обычно решают приближенным поиском в векторном индексе. Надежной стратегией является извлечение сразу большого количества кандидатов (скажем, тысяча), которое точно не поместится на экране. Второй этап обычно более гранулярный, более проработанный и искусный, он формирует финальную выдачу, отбирая среди множества кандидатов 5-10 самых релевантных предметов, которые достойны того, чтобы оказаться на начальной странице приложения.

Отбор кандидатов

Поиск по контенту

Представим ситуацию. Вы руководите проектом по разработке социальной сети для обмена фотографиями. Каким образом можно формировать ленту рекомендаций для пользователя?

В этой гипотетической ситуации одно из самых простых решений — предлагать тот контент, который похож на контент, с которым пользователь ранее взаимодействовал. То есть мы можем сделать следующее:

- обратиться к истории лайков пользователя и взять несколько последних фотографий как за образец
- найти на нашей площадке topk фотографий, похожих на эти образцы

Отбор кандидатов свёлся к обращению к поисковому движку. Методы поиска хорошо развиты в наши дни. Можно осуществлять поиск по текстам, веб-страницам, картинкам.

Выделим плюсы и минусы такого подбора кандидатов. Плюсы:

- такую систему легко масштабировать на новых пользователей, поскольку она не требует никакой информации кроме истории взаимодействий
- мы моделируем рекомендации напрямую исходя из предпочтений пользователя

Минусы:

- мы не предлагаем пользователю ничего нового
- не учитываем информацию о пользователе, о взаимодействии и проч

Так работают разделы рекомендаций наподобие "Похоже на просмотренное ранее".

Колаборативная фильтрация

Одним из простейших способов решить проблему новых рекомендаций — колаборативная фильтрация. Под эту парадигму подпадает очень широкий набор методов. Дать определение в паре слов достаточно трудно, поэтому быстренько вспомним конкретные примеры: k nearest neighbors и matrix factorization.

KNN

Данный метод исходит из простого предположения: если у двух пользователей похожая история взаимодействий с предметами, то предметы, понравившиеся одному из них, можно рекомендовать второму.

Эта идея обычно реализуется с помощью так называемой матрицы взаимодействий. Это матрица $R \in \mathrm{Mat}(n_u \times n_i)$, где n_u — количество пользователей в системе, n_i — количество предметов.

Тогда строка матрицы R_u, соответствующая пользователю под номером u, является своего рода векторизацией. Чтобы подобрать рекомендации для пользователя u, достаточно найти k ближайших к нему пользователей $N(u) = \{u_1, \ldots, u_k\}$ и посмотреть, какие товары есть в их истории взаимодействия. Формально говоря, нужно выбрать top-k предметов по предсказанной релевантности:

$$\widehat{R}_{ui} = \frac{1}{|N(u)|} \sum_{\tilde{u} \in N(u)} R_{\tilde{u}i}.$$

Так работают рекомендации в духе "С этим также слушают".

MF

Если говорить грубо, то колаборативная фильтрация методом matrix factorization похожа на фильтрацию методом KNN, в котором KNN-предсказания заменили на линейную регрессию.

Если говорить строго, то метод заключается в обучении эмбедингов для пользователей и предметов. Инициализируем матрицу $P \in \mathbb{R}^{n_u \times d}$, одна строчка p_u соответствует эмбедингу пользователя под номером u. Аналогично, предмету i будет соответствовать строчка q_i в матрице $Q \in \mathbb{R}^{n_i \times d}$.

Мерой релевантности выступит следующее:

$$\widehat{R}_{ui} = \langle p_u, q_i \rangle + b_u + b_i + \mu.$$

Скаляры b_u, b_i, μ выступают в роли bias'a. Обучаются такие эмбединги на MSE-loss.

По итогу эмбединги формируют кластеры похожих пользователей и предметов, поскольку выполнено важное свойство:

$$\widehat{R}_{ui} \sim \langle p_u, q_i \rangle.$$

Плюсы и минусы KNN и MF

Плюсы:

- мы рекомендуем что-то совершенно новое!
- очень легко обучить, поскольку для рекомендаций требуется всего лишь матрица взаимодействий и абсолютно никаких знаний о доменной области
- очень легко задеплоить, потому что все опять свелось к построению векторного индекса

Минусы:

- проблема холодного старта:
  - при добавлении нового товара у него пустая история, поэтому чисто математически такой алгоритм никогда не будет его рекомендовать
  - аналогично при добавлении нового пользователя

  Эту проблему необходимо решать путем привлечения других алгоритмов рекомендаций. Новым пользователям можно рекомендовать просто самые популярные предметы. А новые предметы пытаться выдавать на основе сторонних признаков.
- алгоритм никак не учитывает сторонние признаки

Методы KNN и MF могут быть хорошим бейзлайном и стартовой точкой. Но в крупных сервисах они, конечно, уже не используются.

Контрастивное обучение

Идея

Пусть имеются матрицы признаков для пользователей $\mathcal{U} \in \mathbb{R}^{n_u \times f_u}$ и предметов $\mathcal{I} \in \mathbb{R}^{n_i \times f_i}$. Дополнительно мы имеем матрицу взаимодействий $R \in \{0, 1\}^{n_u \times n_i}$. Обучать не эмбединги p_u, q_i для пользователей и товаров, а нейросети $p_\phi(u)$, $q_\theta(i)$, которые принимают на вход признаковые описания $u \in \mathbb{R}^{f_u}$, $i \in \mathbb{R}^{f_i}$ и выплевывают векторные представления:

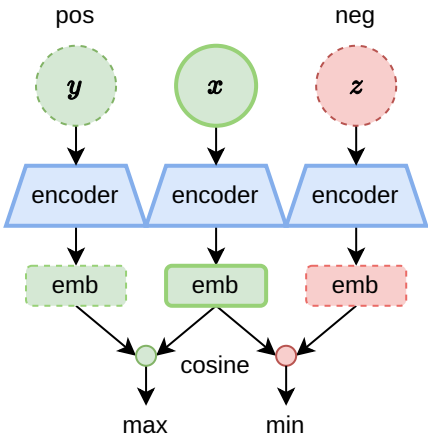
$$\widehat{R}_{ui} \sim \langle p_\phi(u), q_\theta(i) \rangle =: s(u, i).$$

Обучение

Обучают такие нейросети с помощью контрастивного обучения. Напомним, что оно заключается в оптимизации следующего функционала:

$$\mathcal{L}_{ui} = -\log \frac{\exp(s(u,i))}{\exp(s(u,i)) + \sum_z \exp(s(u,z))}.$$

Схематически это можно представить следующим образом:



В роли положительных пар играют пары пользователь-товар которые взаимодействовали. В роли негативных — пары, которые не взаимодействовали.

Обсуждение

Рассмотрим плюсы и минусы. Плюсы:

- предсказываем что-то новое для пользователя
- простота деплоя (опять векторный индекс)
- алгоритм напрямую использует сторонние признаки
- решена проблема холодного старта, поскольку мы избавились от обучаемых матриц эмбедингов

Минусы:

- процедура обучения может быть достаточно сложной, необходимо учесть много гиперпараметров; подробнее можно почитать в [sampling-bias-corrected].
- слабо учитывается учитывается контекст, только в виде временных признаков

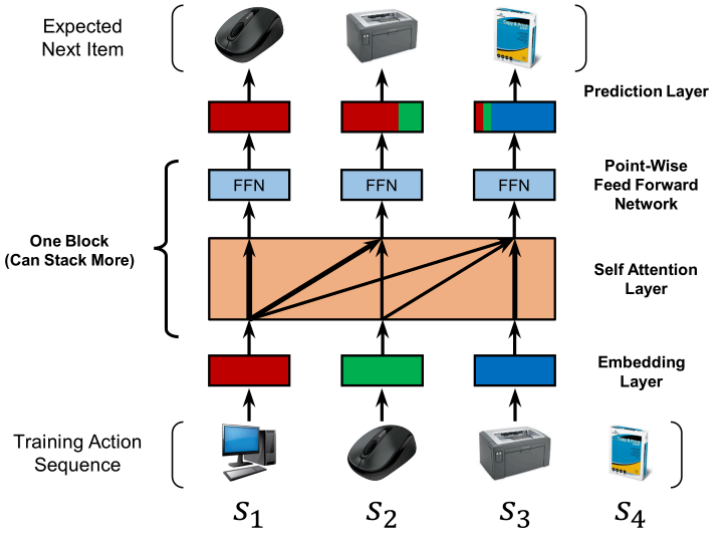
Такие двухбашенные модели уже не называют колаборативной фильтрацией, поскольку здесь нет обучаемых эмбедингов и используется информация помимо колаборативного сигнала.

Sequential Recommenders

Отдельным разделом в рекомендательных системах является sequantial recommenders. Идея в том, чтобы эффективнее использовать информацию не просто об истории взаимодействий в целом, а о последних взаимодействиях или даже о тех, которые были сделаны прямо сейчас. Например, после покупки ноутбука может быть удачно предложить купить аксессуары к нему, а потом аксессуары к аксессуарам и так далее...

Идея простая. Давайте воспринимать предметы, с которыми взаимодействовал пользователь, как последовательность, упорядоченную по времени взаимодействия. Для каждого предмета будем обучать эмбединг. Задача рекомендации сводится к предсказанию следующего товара. Ничего не напоминает? Так это же прямо как казуальное языковое моделирование!

В работе под названием SASRec (Self-attentive sequential recommender) авторам удалось решить такую задачу с помощью GPT-like модели: двухслойный трансформер с маскированным атеншеном, разделение весов между эмбедингами товаров и предсказывающей головой, обучение на кросс-энтропию.



Обратим внимание не алгоритм предсказания в GPT-like моделях. Пусть $h_t \in \mathbb{R}^d$ — финальное скрытое представление предмета, стоящего в последовательности под номером t. Оно подается в классификатор, отображающий h_t в логиты по всем возможным следующим предметам.

$$\mathrm{Softmax}(Wh_t) \in \mathbb{R}^{n_i}.$$

В качестве матрицы $W \in \mathbb{R}^{n_i \times d}$ используют сами эмбединги предметов. А произведение Wh_t есть ни что иное как подсчет скоров между h_t — эмбедингом пользователя, — и эмбедингами всех предметов. Поскольку мы не учитывали вообще никакую информацию, кроме истории взаимодействий, получается что SASRec это метод колаборативной фильтрации :)

На практике SASRec (и его брата BERT4Rec) не называют колаборативной фильтрацией, потому что эту модельку вычислительно сложнее тренировать. При этом говорят, что модель обучена с использованием колаборативного сигнала.

Ранжирование

Постановка задачи

Перейдем наконец к задаче ранжирования. Имеется пользователь u и набор кандидатов $\{i_1, \ldots, i_k\}$. Задача: отсортировать (=отранжировать) кандидатов по релевантности.

Эту задачу, наверное, всегда пытаются решить, предсказывая скор для каждого отдельного кандидата. При обучении ранкера для каждого кандидата есть некоторый золотой скор. Главная проблема состоит в том, чтобы при обучении учесть не только информацию о скоре одного единственного кандидата, а о совокупности кандидатов. Ибо это источник дополнительной информации.

Методы решения

Выделяют несколько подходов к обучению ранкеров.

Pointwise. Поточечные методы никак не учитывают информации о совокупности кандидатов. Вместо этого в тупую учится модель на задачу регрессии или бинарной классификации.

Pairwise. Попарные методы используют информацию о попарных сравнениях кандидатов. Пусть $s_\psi(i)$ — вещественный скор, который ранкер s_θ выдал для кандидата i. Пусть t_i — истинный скор. Обучение обычно сводится к минимизации следующей функции потерь:

$$\mathcal{L}_{ij} = \log(1 + \exp[\{s_\psi(i) - s_\psi(j)\} \cdot \operatorname{sign}\{t_i - t_j\}])$$

Это не что иное как логлосс для меток ± 1.

Listwise. Эти методы пытаются умным образом аккумулировать информацию обо всех кандидатах в одну лосс функцию.

Обсуждение

Часто (по крайней мере в прошлом десятилетии) в качестве s_ψ выступал градиентный бустинг. Самым известным списочным методом ранжирования является LambdaRank. Его реализацию можно до сих пор найти в современных библиотеках для градиентного бустинга.

Сегодня относительно свежим и распространенным методом является DCN-v2 — это pointwise метод, в котором сделан сильный упор на моделирование взаимодействия признаков.