Этап L2

Задача: ресерч проблематики, определение набора подходящих решений
Итог работы: проведен поиск решений среди научных работ/статей, отобраны наиболее оптимальные решения и сформирована итоговая архитектура решения.

Здесь представлены основные идеи нескольких статей по коллаборативной фильтрации и математике, связанной с данным подходом, оказавшиеся крайне полезными для определения пути решения задачи. В предпоследнем разделе представлена архитектура модели, которую буду пытаться реализовать на следующих этапах.

Статья 1. Collaborative Filtering for Implicit Feedback Datasets

Авторы: Yifan Hu, Yehuda Koren, Chris Volinsky
Организация: AT&T Labs **(2008)**
Ссылка: http://yifanhu.net/PUB/cf.pdf

Существует 2 подхода к рекомендациям: Content Based, основанный на свойствах и метаданных юзеров и айтемов, и Collaborative Filtering, основанный на поведении пользователя в прошлом. Последний и рассматривается. В отличие от Content Based подхода, CF страдает от проблемы “холодного старта”.

Данные могут быть явными и неявными. В Задаче 1 мы имеем дело с неявным фидбеком. В таком случае от юзера не требуется никаких дополнительных действий, чтобы данные для модели были собраны. Авторы работали над механизмом рекомендаций ТВ-шоу.

У неявного фидбека выделяется несколько ключевых свойств:
1. *Нет негативного фидбека*. Тот факт, что юзер никогда не взаимодействовал с айтемом не означает, что айтем ему не нравится, но именно в пропущенных данных и будет, скорее всего, храниться информация о том, понравится ли айтем юзеру;
2. *Шумность*. Факт покупки айтема не означает, что айтем юзеру нравится: он мог быть куплен в подарок или не понравиться самому юзеру. Юзер мог полностью просмотреть телепередачу, заснув на предыдущей;
3. *Численное значение явного фидбека отражает предпочтения, однако для неявного фидбека - это уверенность, и является частотой взаимодействия юзера с айтемом*. БОльшее значение не означает предпочтительность айтема (фильм VS скетч), но повторное взаимодействие даёт больше уверенности в наблюдении, чем произошедшее единожды;
4. *Особенность оценки рекомендательной системы*. В традиционном подходе юзер численную оценку, и можно посчитать, например, MSE. В случае неявного фидбека непонятно, как оценивать шоу, просмотренное лишь однажды, или сравнивать 2 шоу, идущие одновременно по разным каналам.

r_{ui} обозначает, сколько раз юзер полностью посмотрел передачу. Как правило, CF основана на моделях соседства, и в оригинальном виде базируется на user2user подходе. Наша цель - предсказать ненаблюдаемое значение r_{ui}. Модели скрытых факторов ставят своей целью выявление скрытых признаков, объясняющих наблюдаемые r_{ui}.

Чтобы перейти от предпочтительности к уверенности, r_{ui} бинаризуется, и переходим к p_{ui}. По мере роста r_{ui} у нас появляются более сильные признаки того, что айтем действительно нравится юзеру. Для отображения уверенности в наблюдении p_{ui} вводится c_{ui}: $c_{ui} = 1 + \alpha r_{ui}$. Скорость возрастания уверенности в наблюдении контролируется константой α. Так, сделали переход: $r_{ui} \rightarrow (p_{ui}, c_{ui})$.

После того, как запишем матрицу R как $R \approx X^T Y$, наша задача — найти вектор $x_u \in R^f$ для каждого юзера u и вектор $y_i \in R^f$ для каждого элемента i, которые будут учитывать пользовательские предпочтения. Тогда предпочтения будут вычисляться как: $p_{ui} = x_u^T y_i$. Векторы отображают юзеров и айтемы в общее пространство скрытых факторов, где их можно будет сравнить.

Введя регуляризацию, получим функцию потерь вида:

$$\sum_{u,i} c_{ui}(r_{ui} - x_u^T y_i)^2 + \lambda(\sum_u ||x_u||^2 + \sum_i ||y_i||^2) \rightarrow min(x_u, y_i)$$

Если обозначить число юзеров за m, число айтемов - за n, то число элементов в матрице взаимодействий равно mn. Это число часто составляет несколько миллиардов, поэтому привычный SGD не применим.

Когда вектор латентных факторов юзера или айтема зафиксирован, функция потерь квадратичная, и становится гораздо проще найти её глобальный минимум. В этом и заключается *Alternating-least-squares* процедура оптимизации: мы чередуем (alternate) пересчёт значений юзер-вектора с айтем-вектором, и каждый такой шаг гарантирует нам снижение значения функции потерь.

Итерационная процедура оптимизации ALS:

Шаг 1. Делаем пересчёт юзер-векторов со скрытыми факторами. Для этого введём матрицу Y размерности $n \times f$. Перед тем, как пробежаться по всем юзерам, предрассчитаем матрицу $Y^T Y$ размерности $f \times f$, т.е. сложность составит $O(f^2 n)$. Для каждого юзера определяем диагональную матрицу весов C^u размерности $n \times n$, состоящую из введённых выше уверенностей: $C^u_{ii} = c_{ui}$. Также определяем вектор предпочтений для каждого юзера $p(u) \in R^n$, т.е. $p(u) = (p_{u1}, \ldots, p_{un})$. В результате дифференцирования получим аналитическое решения для x_u, которое минимизирует функцию потерь:

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p_u.$$

Заметим, что вычисление $Y^T C^u Y$ потребует $O(f^2 n)$ для каждого юзера u. Для ускорения вычислений заметим, что $Y^T C^u Y = Y^T Y + Y^T (C^u - I) Y$. В этом выражении $Y^T Y$ не зависит от u и была уже предрасчитана. Обозначим за n_u количество непустых взаимодействий юзера u и тогда $(C^u - I)$ имеет лишь n_u ненулевых элементов, причём $n_u << n$. При этом $C^u p(u)$ тоже содержит n_u ненулевых элементов. Опустим выкладки относительно сложности вычислений, обозначим за $N = \sum_u n_u$ и получим итоговую сложность вычислений для всех m юзеров: $O(f^2 N + f^3 m)$. **Так, время, требующееся для вычислений, линейно зависит от размера входных данных. Значение f обычно лежит от 20 до 200**.

Шаг 2. Полностью аналогичная логика сохраняется для пересчёта значений айтем-векторов, и тогда:

$$y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p_i.$$

Сложность этого шага равна $O(f^2 N + f^3 n)$

Авторы отмечают, что обычно делается 10 таких итераций. После вычислений векторов латентных факторов для юзеров и айтемов мы рекомендуем юзеру u K доступных товаров с наибольшим значением величины $p_{ui} = x_u^T y_i$ (p с колпачком).

Алгоритм решения можно по-разному модифицировать. В частности, можно по-разному переходить от r_{ui} к p_{ui}. Например, можно использовать порог значения r_{ui}, с которого p_{ui} будет иметь ненулевое значение. Во-вторых, можно по-разному трансформировать r_{ui} к уверенности c_{ui}. У авторов, помимо базового, хорошо сработал следующих подход: $c_{ui} = 1 + \alpha \cdot log(1 + r_{ui}/\epsilon)$. **Это попробуем использовать для модификации базового алгоритма.**

Переход от r_{ui} к 2 величинам p_{ui} и c_{ui} лучше отражает природу данных и обеспечивает повышение точности при прогнозировании.

Ввиду того, что мы абстрагируемся от прошлых действий юзера и переходим к вектору факторов юзера, пропадает прямая взаимосвязь между прошлыми действиями юзера и выданными рекомендациями. Однако ALS-модель обеспечивает интересный подход к объяснению предложенных юзеру рекомендаций. Вспомним, что

$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p_u$ и тогда (с колпачком):

$p_{ui} = x_u^T y_i \;=\; y_i^T x_u = y_i^T (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p_u$.

Обозначим матрицу $(Y^T C^u Y + \lambda I)^{-1}$ размерности $f \times f$ за W^u, которую можно рассматривать как взвешивающую матрицу для юзера u. Взвешенная схожесть между айтемами i и j с точки зрения юзера u, которую обозначим за s_{ij}^u, равна $s_{ij}^u = y_i^T W^u y_j$.

Тогда предсказанное p_{ui} можно переписать как: $p_{ui} = \sum_{j:r_{uj}>0} s_{ij}^u c_{uj}$. В таких обозначениях предпочтения предсказываются как линейная функция от прошлых действий ($r_{uj} > 0$), взвешенных на сходства между айтемами. В этой связи модель матричного разложения с ALS можно рассматривать как (preprocessor) для моделей, основанных на соседстве, где схожесть айтемов рассчитывается с помощью такой особой процедуры оптимизации.

Модель с выведенной выше функцией потерь перебила все другие опробованные модели.

Статья 2. Applications of the Conjugate Gradient Method for Implicit Feedback Collaborative Filtering

Авторы: Gábor Takács, István Pilászy, Domonkos Tikk
Организация: Gravity Research and Development Ltd. **(2011)**
Ссылка: http://rs1.sze.hu/~gtakacs/download/recsys_2011_draft.pdf

Алгоритм CF с неявным фидбеком, как правило, требует решения задачи взвешенной Ridge-регрессии. Зачастую нет достаточного количества времени для вычисления точного решения, поэтому используются приблизительные подходы. В работе сравниваются 2 таких метода: покоординатный спуск и сопряжённый градиент. Покоординатный спуск и является стандартным ALS, и авторы приходят к выводу, что по скорости сходимости он проигрывает методу сопряжённого градиента.

Для ускорения стандартной процедуры оптимизации ALS автор библиотеки, ссылаясь на данную статью, предлагает использовать другой метод получения численного решения – метод сопряжённого градиента (conjugate gradient method → CG). Принцип его работы в туториале создателя библиотеки описан существенно понятнее, чем в оригинальной работе.

В общем смысле CG учитывает значение градиента не только на текущем, но и на предыдущем шаге с весом β.

Возможно, CG и есть momentum для градиентного спуска или бустинга, но в статье такой терминологии нет.

Особое свойство этого метода в том, что направления шага (search directions) выбираются таким образом, что точка оптимума достигалась не более, чем за f шагов. Напомню, что f - это количество латентных фичей, которые будем извлекать из данных при SVD-разложении, и является гиперпараметром.

Изложенный в **Статье 1** и реализованный в **Статье 3** ALS-подход может рассматриваться как частный случай метода сопряжённого градиента, когда $\beta = 0$.

Матричная факторизация предполагает представление матрицы R размерности $N \times M$ в виде произведения двух других матриц меньшей размерности: $R = PQ^T$, где $P \in R^{N\times K}$, $Q \in R^{M\times K}$. Определим множество непустых взаимодействий как T, и тогда аппроксимация делается исключительно по всем $(u, i) \in T$.

Функция потерь (least squares cost function) для неявного фидбека определяется как:
$g(P, Q) = \sum_{(u,i)\in T} c_{ui}(r_{ui}^{est.} - r_{ui})^2 + \lambda_P||p_u||^2 + \lambda_Q||q_i||^2$.
Для всех ячеек, кроме непустых, c_{ui} и r_{ui} константы.

Функция g - невыпуклая, поэтому точная минимизация крайне затруднительна. Здесь на помощь приходит ALS, предлагающий поочерёдно шагать по P и по Q. В общем виде в оригинальной работе по ALS в результате дифференцирования получаем: $p_u = (Q^TC_uQ + n_u\lambda_PI)^{-1}(Q^TC_ur_u)$. Такая же логика и для q_i. Такой вариант считается “наивным” и в той же оригинальной работе приводится модификация.

Обозначим за $u = 0$ юзера, у которого совсем нет (положительного) фидбека, т.е. он не взаимодействовал ни с одним айтемом. Предрассчитаем для него Q^TC_0Q и $Q^TC_0r_0$. Заметим, что для любого другого произвольного юзера u разница между C_0 и C_u будет мала, т.к. юзер взаимодействует лишь с довольно малым подмножеством айтемов. Тогда вместо более вычисления Q^TC_uQ напрямую мы сможем рассчитать его как $Q^TC_0Q + Q^T(C_u - C_0)Q$, т.е. нужно будет учитывать только разницу между юзерами. Это и обосновывает разложение произведения матриц в Статье 1, ведь всем нулевым элементами матрицы взаимодействий присваивается уверенность 1 и тогда совокупность предпочтений может быть описана единичной матрицей I.

Вычисление X_u и Y_i в таком виде является довольно затратным, поэтому с помощью метода CG авторы уходят от такого подхода. В общем виде опишем решение. Изложение получилось смесью Статьи 2 и записей создателя библиотеки implicit.

Вспомним, что $Y^TC^uY + \lambda I = Y^TY + Y^T(C^u - I)Y + \lambda I$. Самым трудоёмким здесь является вычисление Y^TC^uY, чего и избегает автор. Предрассчитаем $Y^TY + \lambda I$. Далее начинается итеративная процедура.

Пробегаем по всем юзерам. Вспомним, что мы бинаризовали исходную матрицу R и в формуле используется бинарное представление предпочтений P_u.

В общем виде эта задача формулируется как линейное уравнение: $Ax = b$. Нам надо восстановить матрицу латентных фичей юзеров X, и в Статье 1 мы в результате дифференцирования получили, что $X_u = (Y^T C_u Y + \lambda I)^{-1} Y^T C_u P_u$.
Тогда в нашем случае $A = Y^T C_u Y + \lambda I$, $b = Y^T C_u P_u$

Далее вводится функцию, которая будет показывать, приближаемся ли мы к оптимуму X_u: $f(x) = \frac{1}{2} x^T A x - x^T b$. Дифференцируем её, получаем градиент: $\nabla_x f(x) = Ax - b$. На основании неё получаем остаток k-го шага: $r_k = b - Ax_k = Y^T C_u P_u - Y^T C_u Y X_u$, т.е. остаток рассчитывается как отрицательный градиент по x_k.

Введём сопряжённые векторы $P = (p_1, \dots, p_f)$. Свойство сопряжённости означает: $p_i^T A p_j = 0$. Тогда матрица P образует базис в пространстве скрытых фичей R^f, в которое мы перешли с помощью MF. Это необходимо, т.к. приближение для X_u будет вида: $X_u^{approx.} = \sum_{i=1}^{f} \alpha_i p_i$. Про α_i поговорим чуть ниже.

Инициализируем $p_0 = r_0$. На каждой итерации будем пересчитывать вектор p_k: $p_k = r_k - \sum_{i<k} \frac{p_i^T A r_k}{p_i^T A p_i} p_i$. За этим шагом стоит сложная математика, в которую придётся углубиться, если буду реализовывать этот алгоритм. p_k задаёт направления шага и, как видно, из формулы, учитывает прошлые направления p_i, $i < k$.

Далее делаем пересчёт на каждом шаге, число шагов - гиперпараметр. Иначе говоря, делаем внутренний цикл. Поэтому все рассчитанные ранее значения имеют индекс не k, а $(k - 1)$ и изначальный вектор $X_u = x_0$.

- $\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$
- $x_k = x_{k-1} + \alpha_k p_{k-1}$
- $r_k = r_{k-1} - a_k A p_{k-1}$
- $p_k = r_k + \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}} p_{k-1}$, причём $\frac{r_k^T r_k}{r_k^T r_k} = \beta$. Обновление p_k делается аналогично градиентному спуску, ведь p_{k-1} – отрицательный градиент, с той разницей, что шаг задаётся не гиперпараметром learning rate, а величиной изменения остатка

На выходе делаем присвоение $X_u = x$. Чтобы не нагромождать, не буду придумывать букву для индекса x и просто отмечу, что это x, полученный на последней итерации внутреннего цикла, который делается заданное число раз, причём не более, чем f.

Покоординатный спуск показал тот же результат, но на каждой эпохе требовалось больше времени для вычислений, а с ростом числа латентных факторов разница между алгоритмами росла, причём немонотонно.

Статья 3. Application of Dimensionality Reduction in Recommender System - A Case Study

Авторы: Badrul M. Sarwar, George Karypis, Joseph A. Konstan, John T. Ried
Организация: GroupLens Research Group, University of Minnesota **(2000)**
Ссылка: http://files.grouplens.org/papers/webKDD00.pdf

CF хорошо зарекомендовала себя, но проблема в том, что с ростом числа юзеров и айтемов базовые алгоритмы начинают плохо себя показывать в значении сравнительно меньшего качества и трудоёмкости процесса вычислений. В работе представлены 2 эксперимента, основанные на SVD-разложении для понижения размерности и построенные на явном фидбеке. Сама же задача построения рекомендаций является частным случаем Knowledge Discovery in Databases.

Подходы, основанные на соседстве (NNA - Nearest Neighbors Algorithm), имеют несколько ограничений:

- *Разреженность данных*. NNA ищут точные совпадения между юзерами/айтемами. Из-за редкости точных совпадений применение NNA приводит к сравнительно невысокому охвату и точности. Корреляция может быть посчитана между юзерами, у которых совпали взгляды по меньшей мере по 2 айтемам, а большое число пар юзеров не будут иметь никакой корреляции;
- *Масштабируемость*. Сложность вычислений сравнительно быстро возрастает с ростом объёма входных данных;
- *Синонимия айтемов*. NNA не способен обнаружить скрытую взаимосвязь, когда (почти) одинаковые айтемы названы по-разному, и будет воспринимать их как 2 совершенно разных айтема.

Подход, использующий для понижения размерности исходной матрицы взаимодействий SVD-разложение, называется Latent Semantic Indexing.

Размерность f векторов скрытых свойств должна быть достаточной, чтобы уловить все важные взаимосвязи в данных, и достаточно малой, чтобы не переобучиться.

Итоговая архитектура решения

Почему матричная факторизация?
Как факторизовать пространство?

[Как заполнять пропущенные значения исходной матрицы?](#)
[Как можно попробовать доработать модель?](#)

Основой будет подход матричной факторизации и оптимизации при помощи ALS, описанный в **Статье 1**. В **Статье 3** обосновывается, что алгоритмы CF, основанные на соседстве, проигрывают MF и по качеству, и по времени вычисления. В частности, MF позволяет не только значительно уменьшить размерность пространства, что сказывается на времени вычислений, но и выявить скрытые свойства юзеров и айтемов, недоступные для других методов CF.

Подбор наилучшего приближения для исходной матрицы R, иначе говоря, оптимизация, будет осуществляться с помощью Alternating Least Squares.

После воплощения данного алгоритма попробую модифицировать исходную модель. Мне видится 3 модификации, которые могут сказаться на качестве.

Модификация перехода к уверенности. В базовом виде уверенность вычисляется по формуле: $c_{ui} = 1 + \alpha \cdot r_{ui}$, но сами авторы метода отмечали, что она может быть по-разному модифицирована. Так, в противовес базовому авторы также предлагают вариант: $c_{ui} = 1 + \alpha \cdot log(1 + r_{ui}/\epsilon)$. Поверхностно сложно предположить, как это отразится на общепринятых метриках, ведь авторы использовали собственную.

Возможно, полезным в значении повышения качества модели было бы учесть в таргете дополнительную информацию

Модификация таргета №1. "Любовь" юзера u к товару i. Можно попробовать учесть, насколько большую долю в общем объёме покупок всех пользователей товара i,,

$\sum_{u} \sum_{t \in T_{u,i}} r_{u,i,t}$, занимают покупки пользователя u, $\sum_{t \in T_{u,i}} r_{u,i,t}$. Чем уникальнее товар i и чем больше спрос юзера к этому уникальному товару, тем больше должно быть значение множителя, т.е. нужна немонотонная функция. Если остановить свой выбор на отношении логарифмов, то выражение для таргета домножится на $\frac{log(\sum_{t \in T_{u,i}} r_{u,i,t})}{log(\sum_{u} \sum_{t \in T_{u,i}} r_{u,i,t})}$.

Модификация таргета №2. Чем позже был совершён заказ, тем более актуальную информацию о предпочтениях пользователя он отображает, отчего более позднему заказу необходимо придать вес бОльший, чем веса более ранних заказов. Каждое из слагаемых, входящее в сумму в ячейке $(u, i) \in R$, т.е. слагаемое, образующее число взаимодействий r_{ui}, можно дисконтировать на функцию, значение которой растёт с увеличением лага между концом промежутка (31.03.2023) и датой t заказа $r_{u,i,t}$. Например, подобрать функцию, подобную упомянутой на [Хабре](#).

Таким образом, бейзлайн-модель - стандартная MF с ALS. 3 модели, которыми будем пытаться превзойти бейзлайн, будут учитывать модификации таргета и уверенности.

На этапе кросс-валидации необходимо будет подобрать несколько параметров:

- f - количество скрытых свойств товара и предпочтений юзера, которые будут использованы при разложении;
- α - коэффициент во втором слагаемом весов c_{ui}, отвечающий за скорость роста уверенности в предпочтении юзером u товара i;
- λ - коэффициент регуляризации. Возможно, следует попробовать регуляризовать с разными коэффициентами отдельно для норм юзер- и айтем-векторов, и тогда подбирать будет надо λ и μ.