Санкт-Петербургский государственный университет
Кафедра математической теории игр и статистических решений

Казаков Данил Игоревич

Магистерская диссертация

Применение гибридных подходов в разработке рекомендательных систем

Направление 010402
Магистерская программа «Исследование операций и системный анализ»

Научный руководитель,
доктор технических наук,
профессор
Буре В.М.

Санкт-Петербург
2019

Содержание

Введение 2
Постановка задачи 5
Обзор литературы 6
Глава 1. Коллаборативная фильтрация 7
1.1 Фукции оценки качества ранжирования 7
1.2 Матричное разложение 8
1.3 Факторизационные машины 9
1.4 Модель LightFM 11
1.5 Обучение ранжированию 12
Глава 2. Контентная модель 15
2.1 Деревья принятия решений 15
2.2 Градиентный бустинг в задаче рекомендации 18
Глава 3. Гибридная рекомендательная система 22
3.1 Гибридизация 22
3.2 Набор данных 23
3.3 Построение решения задачи рекомендации 24
3.4 Результаты 27
Выводы 29
Заключение 30
Список литературы 31
Приложение 33

Введение

На сегодняшний день количество информации и сервисов, предоставлющих её, стремительно растут. И пользователь сталкивается с проблемой выбора релевантной для него информации. Эту задачу и решают рекомендательные системы.

Определение. *Рекомендательные системы* – одно из приложений машинного обучения, задачей которой является предоставление пользователю рекомендаций относительно товаров, которые могли бы ему понравиться.

Приведем несколько примеров рекомендательных систем из разных областей:

- Видеостриминговые сервисы: Netflix, YouTube.
- Музыкальные сервисы: Spotify, Apple Music.
- Новостные сайты: BuzzFeed.
- Социальные сети: Facebook.

Большинство данных сервисов становятся популярными именно благодаря системам рекомендаций. Например, музыкальный сервис Spotify [1] каждый день предлагает множество персонализированных подборок каждый день.

Наиболее популярными являются следующие 2 класса рекомендательных систем:

- Ориентированные на контент. Такие системы ориентируются на характеристики объектов и профиле пользователя.
- Коллаборативная фильтрация. Данный подход учитывает только оценки пользователей относительно объектов, с которыми пользователь уже провзаимодействовал. Основное предположение состоит в следующем: пользователи, которые одинаково оценивали какие-либо объекты, будут давать похожие оценки другим предметам в будущем.

В коллаборативной фильтрации также различают 2 типа оценок пользователя объекту:

[1] https://www.spotify.com

- Явная обратная связь. Пользователь явно сообщает свое мнение относительно объекта, в виде, например, рейтинга. Рейтинги бывают либо бинарными (нравится/не нравится), либо в выраженными в некоторой шкале (например, от одной до пяти звёзд).
- Неявная обратная связь. В данном случае пользователь не сообщает явно свое предпочтение, но при этом система логирует взамодействие пользователя и объекта. Например, человек может полностью посмотреть фильм несколько раз, но при этом явно не сообщать нравится ли ему данный фильм. И система может считать данное взаимодействие положительным.

Минусами неявной обратной связи можно считать тот факт, что мы можем лишь предполагать об истинных предпочтениях пользователя. С другой стороны, неявных откликов намного больше, так как не требуют ничего от пользователя.

В данной работе будут изучены ранжирующие алгоритмы разных типов и все подходы будут исследованы в рамках данных, предоставленным одним онлайн-кинотеатром в рамках соревнования по машинному обучению [15]. Организаторами соревнования были предоставлены данные по просмотрам, проставлениям рейтингов, добавления в избранное фильмов и сериалов. По данным необходимо построить рекомендательную систему и предсказать 20 наиболее релевантных фильмов для каждого пользователя. Функции оценки качества предсказаний будут рассмотрены ниже. Именно решение данной задачи и программная реализация являются основными аспектами данной работы.

Работа состоит из трех глав. В первой главе рассматривается коллаборативная фильтрация. Также изучается обобщение данного подхода – модель факторизационных машин, и конкретная реализация факторизационной модели LightFM [5]. Также рассматривается техника обучения ранжирования (англ. learning to rank). В качестве функции потерь изучается WARP [6].

Во второй главе рассматривается модель деревьев принятия решений и алгоритм градиентного бустинга над деревьями решений. В качестве функции потерь изучается функция LambdaRank [11].

В третьей главе предлагается архитектура гибридной двухуровнен-

вой рекомендательной системы на основе факторизационных машин и градиентного бустинга над деревьями решений. Также приводится подробной описание данной архитектуры, изучается структура данных [15], и приводится результат работы данной системы на действительных данных.

Постановка задачи

Сформулируем задачу данной работы: по данным взаимодействия пользователей составить модель прогнозирования некоторых объектов для каждого пользователя и применить их к действительным данным. Данные относятся к сфере киносмотрения.

Целью данной работы является:

- Изучить модели матричной факторизации и факторизационных машин.
- Исследовать модель градиентного бустинга.
- Изучить модель обучения ранжированию.
- Программно реализовать двухуровневую гибридную рекомендательную систему.
- Применить гибридную рекомендательную систему к реальным данным и получить оценку качества модели.

Обзор литературы

В работе [1] рассматриваются различные гибридные архитектуры рекомендательных систем. В статье [2] рассматриваются основные подходы матричной факторизации. В учебнике [3] обозреваются фундаментальные алгоритмы машинного обучения, и в частности модель деревьев принятия решения, и техники обучения алгоритмов машинного обучения с помощью данных. Также вводятся такие важные понятния, как перекрёстная проверка, регуляризация при обучении и т.д.

В статье [4] впервые вводится понятие факторизационных машин, и формализуется модель зависимости отклика пользователя на некоторый объект черех контекстные признаки. В работе [5] рассматривается модификация факторизационных машин для уменьшения сложности модели. С помощью данной модификации авторы решают проблему холодного старта в задаче рекомендации. Также авторы предоставляют программную реализацию данного алгоритма. В статье [6] в рамках алгоритмов матричной факторизации рассматривается функция потерь WARP [6].

В работе [7] автор Фридман рассматривает ансамбль алгоритмов машинного обучения и предлагает технику градиентного бустинга над деревьями решений. В статье [8] приводится эффективная модификация градиентного бустинга LightGBM. Также авторы предоставляют программную реализацию этого алгоритма.

В ряде работ [9], [10], [11], [12] рассматриваются методы градиентного бустинга в рамках задачи построения рекомендательной системы. В данных работах приводится несколько функций потерь. Одной из них является функция LambdaRank, которая применяется при решение задачи данной работы.

В статье [13] рассматривается метод под названием TF-IDF, данный метод будет использован при расчете признаков контентной модели. В учебнике [14] исследуется метод градиентного спуска. Данный метод довольно прост и используется для обучения всех алгоритмов данной работы.

На странице конкурса [15] приводится описание соревнования рекомендации фильмов для одного онлайн-кинотеатра. В публичном репозитории [16] содержится полная программная реализация гибридной архитектуры рекомендательной системы.

Глава 1. Коллаборативная фильтрация

1.1 Фукции оценки качества ранжирования

Для оценки качества того или иной рекомендательной системы нам необходимо ввести функции оценки качества. Пусть U – множество пользователей, I – множество объектов. Результат работы алгоритма рекомендации для каждого $u \in U$ – это отображение $r : I \to \mathbb{R}$, где $r_u(i)$ характеризует степень релевантности объекта $i \in I$ пользователю $u \in U$.

Отображение r задает перестановку $\pi : [1..m] \to [1..m]$, где $m = |I|$. Под $\pi^{-1}(k)$ будем понимать элемент i, который в результате перестановки π оказался на k-ой позиции. Зададим индкиаторную функцию r^{true} – истинную функцию релевантности, которая равна 1, если объект релевантен данному пользователю, и 0 в противном случае.

Тогда под точностью на K (precision at K) элементах будем считать:

$$p@K = \frac{\sum_{k=1}^{K} r^{true}\left(\pi^{-1}(k)\right)}{K}$$

Под полнотой на K элементах (recall at K) будем понимать следующее выражение:

$$r@K = \frac{\sum_{k=1}^{K} r^{true}\left(\pi^{-1}(k)\right)}{n_u}, \tag{1}$$

где n_u – количество просмотренных элементов пользователем u за рассматриваемый период.

Недостаток функции $p@K$ в том, что она не учитывает порядок рекомендуемых элементов. Например, если из K элементов для пользователя релевантен лишь один объект, то неважно на какой позиции этот находится объект.

Под усредненной точность на K элементах (average precision at K) будем счиать:

$$ap_u@K = \frac{1}{K}\sum_{k=1}^{K} r^{true}\left(\pi^{-1}(k)\right) \cdot p@k$$

Данная функция качества уже чувствительна к порядку рекомендуемых элементов. Например, если из трех элементов релевантным оказался

элемент на 3 позиции, то функция $ap@3$ будет равна $\frac{1}{9}$, а если релевантный элемент окажется на первом месте, то $ap@3$ будет равна $\frac{1}{3}$.

Отметим, что $ap@K$ считается только для одного пользователя. Для всех $u \in U$ можно посчитать следующую величину:

$$map@K = \frac{1}{N} \sum_{u \in U} ap_u@K,$$

где $N = |U|$.

В общем случае для каждого пользователя количество релевантных элементов n_u может оказаться меньше, чем K. Тогда можно скорректировать метрику $ap@K$ следующим образом:

$$nap_u@K = \frac{1}{min(K, n_u)} \sum_{k=1}^{K} r^{true}\left(\pi^{-1}(k)\right) \cdot p@k$$

Соответственно:

$$mnap@K = \frac{1}{N} \sum_{u \in U} nap_u@K \tag{2}$$

С помощью функции $mnap@K$ (3) мы в дальнейшем будем оценивать качество ранжирования исследуемых систем.

1.2 Матричное разложение

Ещё раз отметим, что результат работы ранжирующего алгоритма для каждого $u \in U$ – это отображение $r : I \to \mathbb{R}$, где $r_u(i)$ характеризует степень релевантности объекта $i \in I$ пользователю $u \in U$.

Определение. *Матричная факторизация* – это класс алгоритмов коллаборативной фильтрации, используемых в рекомендательных системах. Алгоритмы матричной факторизации работают путем разложения матрицы взаимодействия пользователь-элемент на произведение двух прямоугольных матриц меньшей размерности. [2]

Пусть у нас есть данные D взаимодействий (оценок) пользователей $u \in U$ и объектов $i \in I$. По всем взаимодействиям мы можем построить матрицу пользователь-объект R размерностью $n \times m$, где $n = |U|, m = |I|$.

Наша задача заключается в том, чтобы матрицу R размерности $n \times$

m представить в виде произведения двух матриц H размерностью $n \times k$, и W размерностью $k \times m$, где k – размерность пространства векторов пользователей и объектов. То есть матрица предсказанных оценок $\hat{R}$ будет представима в виде:

$$\hat{R} = H \cdot W$$

Другими словами, мы хотим для каждого пользователя $u \in U$ и элемента $i \in I$ получить векторное представление в пространстве R^k таким образом, чтобы $\hat{r}_{ui} = \sum_{j=1}^{k} v_{vk} \cdot w_{ki}$.

Для получения матриц H и W мы можем получить следующим образом:

$$\arg\min_{H,W} \|R - \hat{R}\|$$

В качестве нормы матрицы обычно выбирается эвклидова норма.

1.3 Факторизационные машины

Факторизационные машины являются в некотором смысле обобщением матричной факторизации. Помимо матрицы взаимодействий пользователей и объектов в модель включаются признаки пользователей и объектов. Такой подход рассматривается в статье [34.

Пусть у нас есть датасет $D = \left\{ \left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \dots \right\}$ примеров взаимодействия пользователей и объектов, $x^{(i)} \in \mathbb{R}^n, y^{(i)} \in \mathbb{R}$. Вектор y означает отклик взаимодействий x.

Вектор $x^{(i)}$ представляет собой признаки взаимодейтвия. В случае матричной факторизации это были бы признак определенного пользователя и признак определенного объекта. В случае факторизационных машин признаковое пространство нескольно шире.

Для примера возьмем задачу рекомендации фильма. Каждый фильм, который мы рекомендуем, может характеризоваться такими признаками, как жанр, страна производства, режиссер и т.п. С другой стороны, профиль пользователя может быть представлен возрастом, полом, регионом проживания и т.п. Данные признаки несут важную информацию и крайне важны для построения ранжирующего алгоритма.

Отметим, что такие признаки, как признак определенного пользо-

вателя или признак конкретного объекта мы будем строить с помощью следующего алгоритма. В общем случае пусть у нас есть множество A, и для каждого $a \in A$ нам необходимо построить признаковое описание.

- Сопоставим каждому элементу $a \in A$ натуральное число от 1 до N, где N – мощность множеста A. Нумеровать мы можем каким угодно способом. Пусть n_a – порядковый номер элемента a.
- Для каждого $a \in A$ построим вектор длины N, все координаты которго будут равны 0, кроме координаты под номером n_a, она будет равна 1.

Теперь рассмотрим модель факторизационных машин:

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle \, x_i x_j, \tag{3}$$

где параметры модели определим следующим образом:

$$w_0 \in \mathbb{R}, \quad \mathbf{w} \in \mathbb{R}^n, \quad \mathbf{V} \in \mathbb{R}^{n \times k},$$

где n – размерность признаково пространство, k – размерность пространства векторов, сопоставленных каждым признакам.

Под $\langle \cdot, \cdot \rangle$ мы будем понимать векторное произведение векторов:

$$\langle \mathbf{v}_i, \mathbf{v}_j \rangle = \sum_{f=1}^{k} v_{i,f} \cdot v_{j,f}$$

Отметим параметры модели, которые мы хотим найти, исходя из полученных данных:

- w_0 – глобальное смещение модели
- w_i – вес i-го признака
- $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ – вес взаимодействия i-го и j-го признаков.

Так как функция (3) не выпукла, то мы не можем найти аналитическое решение. В таких случаях обычно применяют метод градиентного спуска, который описан в [14].

Определение. Градиентный спуск — метод поиска локального экстремума функции с помощью движения вдоль градиента.

Мы не будем здесь приводить вычисления градиента, а лишь вынесем результаты. Пусть θ – какой либо параметр модели (3), Тогда

$$\frac{\partial}{\partial\theta}\hat{y}(\mathbf{x}) = \begin{cases} 1, & \text{если } \theta \text{ есть } w_0 \\ x_i, & \text{если } \theta \text{ есть } w_i \\ x_i \sum_{j=1}^{n} v_{j,f} x_j - v_{i,f} x_i^2, & \text{если } \theta \text{ есть } v_{i,f} \end{cases}$$

1.4 Модель LightFM

Модификацие факторизационных машин является модель, предложенная авторами статьи [5]. Вместо того, чтобы учитывать все попарные взаимодествия признаков, авторы предлагают учитывать только вектора пользователей и объектов. При этом эти вектора будут представлены как сумма векторов соответственных признаков.

Пусть f_u – множество номеров признаков, которые относятся к пользователям, f_u – множество номеров признаков, которые относятся к объектам. Тогда вектор пользователя q_u можно представить следующим образом:

$$q_u = \sum_{j \in f_u} \mathbf{v}_j,$$

а вектор объекта

$$p_i = \sum_{j \in f_i} \mathbf{v}_i,$$

Такая же техника применяется и относительно параметров смещения. Для смещения пользователя это будет выглядеть следующим образом:

$$b_u = \sum_{j \in f_u} b_j \tag{4}$$

Соответственно, для смещения элемента:

$$b_i = \sum_{j \in f_i} b_j \tag{5}$$

Модель, которую предлагают авторы в статье [5], выглядит следующим образом:

$$\hat{y}(u, i) = f\left(q_u \cdot p_i + b_u + b_i\right), \tag{6}$$

где

$$b_u, b_i \in \mathbb{R}, \quad q_u, p_i \in \mathbb{R}^k.$$

Функция f может быть выбрана в зависимости от постановки задачи.

Приемуществом модели LightFM, и в целом факторизационных машин, является решение проблемы холодного старта. Именно с помощью модели LightFM авторы [5] решают данную проблему.

Определение. *Проблема холодного старта* – невозможность ранжирующего алгоритма предоставить рекомендации новому пользователю.

Если мы будем знать про нового пользователя его признаки (например: возраст, пол, регион проживания), то факторизационные машины способны посчитать рекомендации по данным признакам. К сожалению, в данной работе не рассматривается проблема холодного старта.

Также особую популярность модель LightFM получил, благодаря программной реализации на языке Python.

1.5 Обучение ранжированию

В параграфе 1.2 мы упомянули, что получить векторные представления пользователей и элементов можно с помощью минимизации среднеквадратичного отклонения матрицы $\hat{R}$ от исходной матрицы R.

На самом деле нам не важна сама оценка, которую пользователь поставит элементу, нам важно для каждого пользователя $u \in U$ построить отношение порядка на множестве элементов I. Другими словами, мы хотим для каждой тройки (u, i_1, i_2), где $i_1, i_2 \in I$ уметь сообщать, что один элемент является более предпочтительным для пользователя, чем другой.

Определение. *Обучение ранжированию* [1] — это класс задач машинного обучения с учителем, заключающихся в автоматическом подборе ранжирующей модели по обучающей выборке, состоящей из множества списков и заданных частичных порядков на элементах внутри каждого списка.

В постановке задачи рекомендации в рамках обучения ранжированию нам необходимы не численные оценки пользователей объектам, а для

каждого пользователя список заданных частичных порядков на множестве объектов I.

Различают несколько типов обучения ранжированию:

Попарный подход. При таком подходе задача сводится к построению бинарного классификатора, которому на вход для каждого пользователя $u \in U$ и для каждой пары $i_1, i_2 \in I$ требуется определить, какой элемент предпочтительней для пользователя u. **Попарный подход.** При таком подходе задача сводится к построению алгоритма, который для каждого пользователя $u \in U$ и списка все элементов $i \in I$ выдает перестановку на множестве элементов. Параметры модели могут настраиваться при максимизации метрик ранжирования, описанных в параграфе 1.1.

Для нашей задачи рекомендации фильмов мы остановились на Weighted Approximate-Rank Pairwise, который описан в статье [6]. В данной работе мы изложим основную идею и результаты данный статьи.

Пусть у нас есть датасет D, состоящий их пар вида (u, y) примеров взаимодействия пользователей и объектов, где элемент y наиболее релевантный объект для пользователя взаимодействия u. Пусть у нас также есть вектор функций $f(u) \in \mathbb{R}^m$, предоставляющих для каждого элемента некоторое значение, где $f_i(x)$ – значение для i-го элемента.

Определим функцию ошибки ранжирующего алгоритма:

$$err(f(u), y) = L\left(rank_y(f(u))\right),$$

где $rank_y(f(u))$ – ранк элемента y, который задается формулой:

$$rank_y(f(u)) = \sum_{i \neq y} (f_i(u) \geq f_y(u))$$

Здесь означает индикаторную функцию. В качестве фунции L авторы предлагают следующий подход:

$$L(k) = \sum_{l=1}^{k} \frac{1}{l}.$$

После ряда преобразований и допущений авторы [6] приходят к приближенной оценке функции $err(f(u), y)$:

$$\overline{err}(f(u), y) = \sum_{i \neq y} L\left(rank_y^1(f(u))\right) \frac{|1 - f_y(u) + f_i(u)|_+}{rank_y^1(f(u))}, \tag{7}$$

где $|t|_+ = max(t, 0)$, а $rank_y^1(f(u))$ принимает вид:

$$rank_y^1(f(u)) = \sum_{i \neq y} (1 + f_i(u) > f_y(u))$$

Постоянная 1 здесь служит некоторым регуляризатором, чтобы уменьшить чувствительность модели к данным.

Далее авторы предлагают считать ошибку не для всех элементов $i \neq y$, а случайно выбирать $i \in I \setminus \{y\}$ до тех пор, пока мы не найдём элемент, нарущающий отношение порядка. Таким образом мы получаем следующую формулу для $rank_y^1(f(u))$:

$$rank_y^1(f(u)) \approx \left\lfloor \frac{m-1}{p} \right\rfloor,$$

где p – число элементов, которые мы случайно выбирали до тех пор, пока не обнаружили элемент i_p, нарушающий правило отношения порядка.

В нашей работе для построения используется метод факторизационных машин LightFM с функцией потерь (7).

Глава 2. Контентная модель

Многие алгоритмы ранжирования полностью игнорируют контекстную информацию о взаимодействиях пользователей и элементов. В отличии от коллаборативной фильтрации и моделей факторизационных машин, ориентированные на контент модели опираются исключительно на признаки пользователя и элемента. В качестве примера можно рассмотреть систему рекомендации, которая предоставляет различные рекомендации в зависимоти от времени суток. В этом случае пользователи могут иметь различные предпочтения в зависимости от времени суток.

Например, в задаче рекомендации фильмов мы можем охарактеризовать каждый фильм следующими характеристиками: страна производства, жанр, актерский состав, год выпуска и так далее.

Кроме того, для каждой пары $(u, i), u \in U, i \in I$ мы можем построить признак взаимодействия пользователя и элемента. То есть для конкретного элемента по истории взаимодействия пользователя мы можем оценить характер взаимодействия с теми же характеристиками, которые присущи определенному элементу.

2.1 Деревья принятия решений

Деревья решений [3] и градинетный бустинг над деревьями решений являются главными инструментами большинства победителей соревнований по машинному обучению. Данные подходы показывают наилучшее качество в ряде задач. Градиентный бустинг над деревьями решений лежит в основе контентной модели, с помощью который мы будем решать задачу рекомендации фильмов. В данной работе мы опишем основные аспекты градиентного бустинга.

В основе градиентного бустинга лежат деревья принятия решений.

Определение. *Дерево принятия решения* – это алгоритм, который представляет собой правила в иерархической структуре данных, представляющее собой двоичное дерево. Каждому объекту соответствует единственный узел, дающий ответ.

Пусть у нас есть какой-либо признак $x \in \mathbb{R}$. Тогда правило, соответствующее узлу $node_i$ задается как $x < c_i$, где c_i – некоторое пороговое

значение. Другими словами, если по правилу узла i признак объекта $x < c_i$, тогда объект переходит к левому потомку узла, если $x \geqslant c_i$, тогда объект переходит к правилу правому потомку узла. Такая процедура продолжается до тех пор, пока объект не попадет в конечный узел, называемым листом. В листе содержится ответ алгоритма.

Остается вопрос построения двоичного дерева. Опишем этот процесс для задачи регрессии. Пусть у нас есть датасет $D = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}$, $i = 1, 2, \cdots n$, где вектор $x^{(i)} \in \mathbb{R}^k$ – признаки i-го объекта, $y^{(i)} \in \mathbb{R}$ – отклик. Для построения дерева нам необходимо определить функцию загрязненности узла, в задаче регрессии мы можем его задать как

$$L = \sum_{i=1}^{n} (y^{(i)} - \hat{y}),$$

где $\hat{y}$ – среднее значение отклика на данных D.

Пусть мы находимся в первом узле узле $node_1$, L_p – загрязненность текущего узла, L_l – загрязненность левого потомка, L – загрязненность правого потомка. Любое правило вида $x < c_i$ разбивает данные D на два подмножества D_l и D_r. Необходимо найти такое правило, чтобы максимизировать следующую величину:

$$L_{delta} = L_p - L_l - L_r$$

Так как у нас множество D конечно, то мы можем найти конечное число разбиений D на два непересекающихся подмножества. В конечных узлах $leaf_i$ будут находится подмножества примеров $D_{leaf_i} \subset D$. Очевидно, что объединение всех подмножеств D_{leaf_i} для всех i будет равно множеству D. Ответом листа $leaf_i$ мы можем считать среднее значения отклкика y объектов, содержащихся в множестве D_{leaf_i}.

Мы изучили построение дерева принатия решения, и теперь рассмотрим технику градиентного бустинга.

Пусть по данным D необходимо восстановить зависимость типа $y = f(x)$. Восстанавливать его мы будем некоторым приближением $\hat{f}(x)$. Для того, чтобы найти такое приближение, необходимо задать некоторую функцию потерь $L(y, f(x))$. Тогда

$$\hat{f}(x) = \arg\min_{f(x)} L(y, f(x))$$

Так как пространство функций f бесконечномерно, то мы ограничимся параметрическим семейством функций $f(x, \theta), \theta \in \mathbb{R}^d$. Тогда мы сводим задачу к задачи оптимизаций значений параметров:

$$\hat{f}(x) = f(x, \hat{\theta})$$
$$\hat{\theta} = \arg\min_{\theta} L(y, f(x, \theta))$$

Значение параметров θ мы будем вычислять итеративно. Выпишем функцию $L_\theta(\hat{\theta})$. После M итераций получается следующие выражения

$$\hat{\theta} = \sum_{i=1}^{M} \hat{\theta}_i$$
$$L_\theta(\hat{\theta}) = \sum_{i=1}^{N} L\left(y_i, f\left(x_i, \hat{\theta}\right)\right)$$

Минимизировать $L_\theta(\hat{\theta})$ мы можем с помощью градиентного спуска [14].

Теперь подставим вместо параметра θ семейство функций $h(x, \theta)$. Так как мы обусловились решать данную задачу методом градиентного спуска, то нам необходимо также задать шаг градиентного шага $\rho \in \mathbb{R}$ для каждой итерации. Теперь наше приближение $\hat{f}(x)$ на шаге t будет выглядеть следующим образом:

$$\hat{f}(x) = \sum_{i=0}^{t-1} \hat{f}_i(x)$$

Для каждого шага итеративного процесса задача выглядит следующим образом:

$$\begin{gathered}(\rho_t, \theta_t) = \arg\min_{\rho,\theta} L(y, \hat{f}(x) + \rho \cdot h(x, \theta)) \\ \hat{f}_t(x) = \rho_t \cdot h\left(x, \theta_t\right)\end{gathered} \tag{8}$$

Теперь мы можем ввести такое понятие как градиентный бустинг на деревьями решений, который был впервые введен Д.Г. Фридманом в статье [7].

Введем значение градиента r_{it} на шаге t:

$$r_{it} = -\left[\frac{\partial L\left(y^i, f\left(x^i\right)\right)}{\partial f\left(x^i\right)}\right]_{f(x)=\hat{f}(x)}, \quad \text{для } i = 1, \ldots, n \tag{9}$$

Д.Г. Фридман предложил на каждом шаге обучать модели так, чтобы предсказания модели $\hat{f}_t$ были максимально близко к значениям градиента. Формально это выглядит следующим образом:

$$\theta_t = \arg\min_{\theta} \sum_{i=1}^{n} \left(r_{it} - h\left(x_i, \theta\right)\right)^2 \tag{10}$$

После того, как мы нашли $\theta_t \rho_t$:

$$\rho_t = \arg\min_{\rho} \sum_{i=1}^{n} L\left(y_i, \hat{f}\left(x_i\right) + \rho \cdot h\left(x_i, \theta_t\right)\right) \tag{11}$$

Формулы (8)-(11) описывают алгоритм градиентного бустинга. В статье [7] автор предложил в качестве $h(x, \theta)$ использовать дерево принятия решений, которое было описано выше.

2.2 Градиентный бустинг в задаче рекомендации

Градиентный бустинг является универсальным инструментом для решения множества задач, таких как задача регрессии или бинарной классификации.

Необходимо задачу рекомендации описать в терминах градиентного бустинга. Рассмотрим задачу рекомендации как задачу бинарной классификации. Пусть для каждого пользователя $u \in U$ мы можем разбить множество объектов $i \in I$ на два непересекающихся подмножества:

- I_{up} – положительные элементы. Под положительными объектами мы понимаем те объекты, которые релевантны пользователю u.
- I_{un} – негативные элементы. Под негативными элементами мы понимаем те элементы, которые нерелавнтны пользователю u.

Мы сделаем некоторое допущение и будем относить объекты, с которыми пользователь не проивзаимодействовал к I_{un}.

В терминах бинарной классификации наш набор данных D будет иметь отклик y следующего вида:

$$y(u,i) = \begin{cases} 0, & i \in I_{un} \\ 1, & i \in I_{up} \end{cases}$$

Для того, чтобы обучить алгоритм нам необходимо задать функцию потерь L, которую необходимо мнимизировать. Для задачи бинарной классификации функция L примет следующий вид:

$$L(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \cdot \log_e (\hat{y}_i) + (1 - y_i) \cdot \log_e (1 - \hat{y}_i) \right],$$

где $\hat{y} = f(x)$ – предсказанные значения модели, принимающее значение от 0 до 1.

В работе [12] приводят функцию потерь $L(y, \hat{f}(x))$, исходя из из постановки задачи рекомендательной системы. Пусть для определенного пользователя $u \in U$ и любой парой $i, j \in I$ вероятность того, что элемент i более релевантен элементу j будет равна:

$$P_{ij} \equiv \frac{1}{1 + e^{-\sigma(s_i - s_j)}},$$

где $s_i = f(x_i)$, $s_j = f(x_j)$ и x_i – вектор признаков пользователя u и объекта i. Можно заметить, что данная функция принимает значение на полуинтервале $(0; 1]$.

Пусть для данной (i, j) $\overline{P}_{ij}$ – это известная вероятность того, что элемент i релевантнее элемента j. Теперь зададим функцию потерь L следующим образом:

$$L = -\overline{P}_{ij} \log P_{ij} - \left(1 - \overline{P}_{ij}\right) \log \left(1 - P_{ij}\right) \tag{12}$$

Введем ещё одно обозначение. Для данной пары $i, j \in I$ пусть $S_{ij} \in \{0, \pm 1\}$ будет равно 1, если объект i более релевантнее j, -1 если i менее релевантнее j, и 0 в случае, если элементы i и j имеют одну степень релевантности.

Тогда истинную вероятность $\overline{P}_{ij}$ запишем как:

$$\overline{P}_{ij} = \frac{1}{2} \left(1 + S_{ij}\right)$$

Перепишем равенство (12) через S_{ij}:

$$L = \frac{1}{2}(1 - S_{ij})\,\sigma(s_i - s_j) + \log\left(1 + e^{-\sigma(s_i - s_j)}\right) \tag{13}$$

Тогда при $S_{ij} = 1$ потери L будут равны:

$$L = \log\left(1 + e^{-\sigma(s_i - s_j)}\right), \tag{14}$$

когда как при $S_{ij} = -1$:

$$C = \log\left(1 + e^{-\sigma(s_j - s_i)}\right) \tag{15}$$

Также заметим, что

$$\frac{\partial L}{\partial s_i} = \sigma\left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{\sigma(s_i - s_j)}}\right) = -\frac{\partial L}{\partial s_j} \tag{16}$$

Предположим, что наша модель f зависит от параметра θ. Для настройки этих параметров нам необходимо необходимо посчитать градиент.

$$\overset{\frac{\partial L}{\partial \theta}}{= \frac{\partial L}{\partial s_i}\frac{\partial s_i}{\partial \theta} + \frac{\partial L}{\partial s_j}\frac{\partial s_j}{\partial \theta}} = \sigma\left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{\sigma(s_i - s_j)}}\right)\left(\frac{\partial s_i}{\partial \theta} - \frac{\partial s_j}{\partial \theta}\right) = \\ = \lambda_{ij}\left(\frac{\partial s_i}{\partial \theta} - \frac{\partial s_j}{\partial \theta}\right), \tag{17}$$

где λ_{ij} мы зададим как

$$\lambda_{ij} = \frac{\partial C(s_i - s_j)}{\partial s_i} = \sigma\left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{\sigma(s_i - s_j)}}\right) \tag{18}$$

В статье [12] авторы проводили эксперименты и показали, что в задаче ранжирования лучше ввести следующую λ_{ij} (мы приведем её для $S_{ij} = 1$):

$$\lambda_{ij} = \frac{\partial C(s_i - s_j)}{\partial s_i} = \frac{-\sigma}{1 + e^{\sigma(s_i - s_j)}}\,|\Delta_{MAP}| \tag{19}$$

В формуле (19) $|\Delta_{MAP}|$ означает модуль дельты метрики ранжирования MAP (4) при смене мест элементов i и j в отринжированном списке (но при этом все остальные элементы из I зафиксированы). Функция потерь с такой модификацией называется LambdaRank.

С помощью формул (13)-(19) мы будем обучать контентную модель

рекомендации фильмов на основе градиентного бустинга над деревьями решения с LambdaRank [12] в качестве функции оптимизации. Детали обучения и рассмотрение признаков модели будет приведено далее в параграфе 3.3.

Глава 3. Гибридная рекомендательная система

3.1 Гибридизация

Различают несколько типов гибридных архитектур:

- Независимая гибридизация.
- Последовательная гибридизация.

В условиях независимой гибридизации несколько различных алгоритмов ранжирования работают независимо друг от друга. Выходы данных систем комбинируются каким-либо образом, и данная комбинация считается выходом независимой гибридной моделью.

Отметим, что рекомендательные системы различаются по сложности. Под сложностью алгоритмов мы понимаем количество ресурсов, необходимых для обучения алгоритма, а также для подсчета рекомендаций. Например, деревья решений в сравнении с факторизационными машинами являются более сложными моделями и показывают лучшее качество, но при этом алгоритм не способен за приемлемое время посчитать для каждой пары $(u, i), u \in U, i \in I$ результат. Но мы можем сократить для каждого пользователя u количество объектов, выбрав наиболее релевантное подмножество $I_u \subset I$. Такое подмножество I_u можно найти менее сложным алгоритмом. Таким образом мы приходим к последовательной гибридизации.

В условиях последовательной гибридизации всю систему рекомендаций можно разбить на несколько уровней. Каждый уровень представляет собой отдельный алгоритм ранжирования, но при этом каждый уровень k для каждого пользователя u принимает множество кандидатов $I_u^k \subset I$ и отбирает подмножество $I_u^{k+1} \subset I_u^k$. Такой подход позволяет использовать сложные алгоритмы рекомендаций, но при этом использовать их на подмножестве всех объектов.

В рамках данной работы предлагается решать задачу рекомендации фильмов [15] с помощью последовательной гибридизации. А именно предлается использовать двухуровневую архитектуру рекомендательной системы, первый уровень которой будет предствлять модель коллаборативной фильтрации, а модель второго уровня – контнетная модель.

3.2 Набор данных

В качестве данных были взята история киносмотрения пользователей в одном онлайн-кинотеатре [15]. Вся история была взята за 6 месяцев. Участникам необходимо было предсказать какие элементы посмотрят те или иные пользователи за последующие 2 месяца.

Каждый просмотр характеризуется следующими полями:

- user_uid — уникальный идентификатор пользователя.
- element_uid — уникальный идентификатор контента.
- consumption_mode — тип потребления (принимает следующие значения: покупка, аренда или просмотр по подписке).
- ts — время совершения операции.
- watched_time — время просмотра в секундах.
- device_type — тип устроиства, с которого была совершена покупка или начат просмотр по подписк.
- device_manufacturer — фирма-изготовитель устроиства, с которого была совершена покупка или начат просмотр по подписке.

Просмотры ялвяются неявной обратной связью. Также для каждого фильма известна его продолжительность. Для определения релевантности просмотра пользователем фильма предлагается считать долю просмотра фильма, то есть отношение времени просмотра к длительности объекта.

Помимо информации о просмотрах нам также доступна информация об оценках, поставленных пользователями. Каждое проставление оценки характеризуется следующими полями:

- user_uid — уникальный идентификатор пользователя.
- element_uid — уникальный идентификатор контента.
- ts — время проставления рейтинга.
- rating — проставленный пользователем рейтинг, принимает значения от 1 до 10.

Оценки являются информацией с явной обратной связью. Данная информация наиболее важна для системы рекомендации, так как пользователь явно сообщил своё мнение относительно конкретного фильма. Но проблема заключается в том, что данной информации довольно мало относительно информации о просмотрах. Выше мы обусловились понимать долю просмотра фильма как степень релевантности. Теперь нам необходимо определить степень релевантности для рейтинга.

Пусть под $rating(u)$ мы будем понимать среднюю оценку, которую пользователь u проставлял некоторым фильмам. Тогда под релевантостью мы будем считать следующую функцию:

$$rel(u,i) = \begin{cases} 0, & rating(u,i) < rating(u) \\ 1, & rating(u,i) \geqslant rating(u) \end{cases}$$

Таким образом, мы свели отклики пользователей с явной и неявной обратной связью в единую шкалу релевантности.

Пользователи в сервисе имеют возможность добавить контент в избранное. Такие данные также доступны для построения рекомендаций.

Также нам доступна анонимизированная метаинформация о контенте:

- type – тип элемента (фильм, сериал или многосерийный фильм)
- duration – продолжительность элемента. Продолжительность элемента необходима для определения релевантности элементов в данных с неявной обратной связью.
- attributes – множество тегов, которыми характеризуется элемент.
- features – набор из пяти анонимизированных признаков.

3.3 Построение решения задачи рекомендации

В данном параграфе содержится описание архитектуры гибридной системы рекомендации. После ряда экспериментов данная архитектура показала лучшие результаты в рамках соревнования по машинному обучению [15].

В качестве архитектуры гибридной рекомендательной системы предлагается обучить двухуровневую систему:

- **Уровень 1.** Факторизационная машина LightFM [5] с функцией потерь WARP [6] для отбора из всего множества элементов I 400 наиболее релевантных элементов для каждого пользователя.
- **Уровень 2.** Контентная модель на основе реализации LightGBM [8] градиентного бустинга с функцией потерь LambdaRank [12] для выбора из 400 кандидатов 20 наиболее релевантных элементов.

Для обучения необходимо разбить датасет за 6 месяцев на две части: тренировочная часть в первые 4 месяца и валидационная часть длиною в последующие 2 месяца.

Валидационная часть нужна именно для того, чтобы мы могли смоделировать тестовый набор данных, который недоступен для участников и на котором оценивается качество алгоритма рекомендации. Тренировочные данные нужны для обучения модели факторизационных машин, а также для сбора признаков на основе этих данных. Модель LightFM [5] мы будем обучать, максимизируя функцию качества (1), потому что мы заинтересованы в том, чтобы в число кандидатов попало как можно больше элементов, которые посмотрит пользователь за валидационный период.

Далее, при обученной модели на тренировочных данных для каждого пользователя u, который взаимодействовал с сервисом в течение валидационного периода, модель первого уровня считает 400 наиболее релевантных элементов. Подчеркнем, что элементы-кандидаты не должны содержать тех элементов, с которыми пользователь уже провзаимодествовал.

Затем, каждый из этих 400 элементов таргетируется следующим образом: если пользователь посмотрел данный элемент в течение валидационного периода, то мы считаем этот элемент положительным, в противном случае – отрицательным. Таким образом мы получаем отклик y.

Для каждого пользователя u и 400 кандидатов у нас имеются следующие группы признаков (в данной работе мы опишем только основные признаки, всего их более 50):

Признаки пользователя.

- Распределение просмотров по девайсам
- Длительность нахождения в сервисе
- Количество просмотров

- Смещение пользователя b_u (4)
- Норма вектора пользователя q_u
- Средний рейтинг элементов, которые были оценены пользователем

Признаки элемента.

- Тип элемента
- Продолжительность
- Количество просмотров
- Смещение элемента b_i (5)
- Норма вектора элемента p_i
- Средний рейтинг элемента, которые были оценены пользователями

Признаки взаимодействия. Данные признаки считаются для каждой пары пользователь-объект (u, i).

- Векторное произведение вектора пользователя q_u и вектора элемента q_i
- Значение модели первого уровня
- Взаимодействие пользователя u с атрибутами данного элемента i.

Признаки взаимодейтвия пользователя u с атрибутами элемента i считаются с использованием техники TF-IDF [13].

Итого, для каждой пары пользователь-кандидат (u, i) мы имеем вектор признаков x и отклик y. С помощью этого датасета мы будем обучать контентную модель.

Имея обученные модели первого и второго уровня мы можем посчитать рекомендации для пользователей тестового периода.

Для изучения данных и реализации алгоритмов ранжирования использовался язык Python, в связи простотой кода, быстрой реализацией и наличием подходящих к решению задачи библиотек:

- NumPy [2]
- Pandas [3]
- LightFM [4]

[2] https://www.numpy.org/
[3] https://pandas.pydata.org/
[4] https://github.com/lyst/lightfm

- LightGBM [5]

Основным вкладом в данной работе считаем именно программную реализацию гибридной архитектуры рекомендательной системы и получения неплохого качества данных рекомендаций в рамках соревнования [15]. Полная программная реализация содержится в [16], в приложении будут содержаться основные функции кода. Также многие нюансы построения гибридной системы описаны в [16] и не рассматриваются в данной работе.

3.4 Результаты

Качество алгоритмов мы будем оценивать с помощью функции оценки $mnap@20$ (2), $k = 20$ выбрано организаторами соревнования [15].

В первую очередь мы должны исследовать релуьтат одноуровневой модели, которая состоит только из факторизационной машины.

Рассмотрим зависимость качества рекомендаций от размерности скрытого пространства $\mathbb{R}^k$:

k	30	50	100	150	200
mnap@20	0.0317	0.0319	0.032	0.0319	0.0319

Как мы видим, наилучшим значением параметра k является 100. Покажем зависимость функции $r@20$ (1) в зависимости от количества итераций обучения модели первого уровня.

.

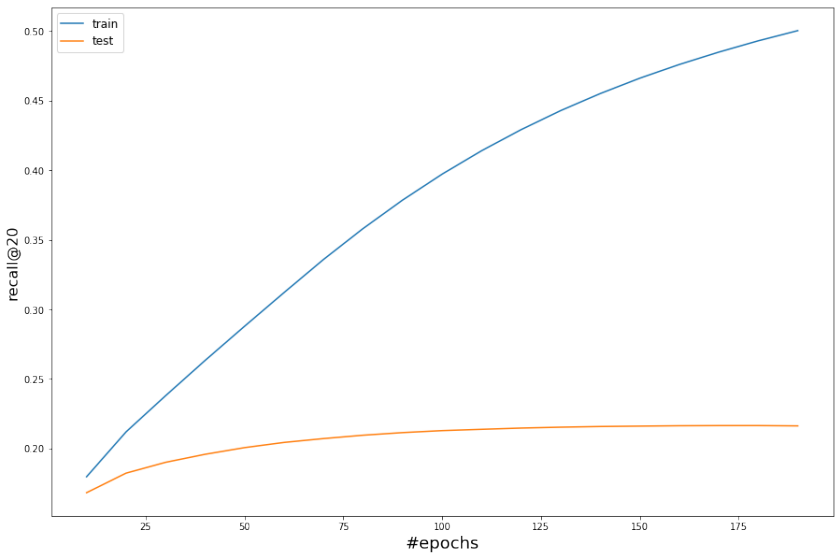

Рис. 1: Зависимость $r@20$ от количества итераций обучения модели.

[5] https://lightgbm.readthedocs.io

Любые попытки использовать дополнительные признаки в модели первого уровня (факторизационные машины) приводили к падению качества. Это можно объяснить лишь структурой самих данных. Таким образом, в нашей задаче модель факторизационных машин представляет собой матричную факторизацию, которая опирается лишь на матрицу взаимодействий пользователей и элементов.

Рассмотрим зависимость качества гибридной модели в зависимости от количества кандидатов c:

c	100	200	400
mnap@20	0.042	0.046	0.049

Отметим наиболее важные признаки (которые привносят наибольший вклад в качество модели):

- Давность выхода элемента в сервисе. В ходе анализа данных было выявлено, что одни из анонимизированных признаков представляет собой дату выхода элемента в сервисе.
- Значение $\hat{y}$ (6) модели LightFM [5].
- Смещение пользователя b_u (4)

Как мы видим, что при увеличении количества кандидатов c возрастает качество гибридной модели, что вполне очевидно. Но для количества кандидатов $c > 400$ возрастает количество ресурсов, необходимых для обучения и предсказания. Поэтому было решено остановиться на количестве кандидатов равных 400.

Также мы обучили контентную модель как решения задачи бинарной классификации (при количестве кандидатов $c = 400$). Значение функции $mnap$@20 оказалась немного ниже – 0.045.

Итого, выведем наилучшие значения $mnap$@20 одноуровневой модели на основе факторизационных машин и двухуровневую модель:

level	1	2
mnap@20	0.032	0.049

Таким образом, при добавлении второго уровня в систему рекомендации мы улучшили функцию $mnap$@20 на 53 процента.

Выводы

В ходе решения задачи построения гибридной рекомендательной системы было рассмотрено и изучено множество алгоритмов рекомендаций. Все алгоритмы были программно реализованы и оценены на реальных данных взаимодествия ползователей в одной онлайн-кинотеатре.

В качестве базовой рекомендательной системы была взята и рассмотрена модель факторизационных машин. Также была расмотрена техника обучения ранжированию и изучена функция потерь WARP [6]. Для решения задачи рекомендации фильмов использовалась реализация LightFM [5]. Такая модель имеет значение функции качества $mnap@20$ равную 0.032 на тестовых данных.

Также была рассмотрена контентная модель в качестве модели второго уровня. Был рассмотрен и изучен алгоритм градиентного бустинга над деревьями принятия решений. В качестве функции потерь в рамках обучения ранжирования был рассмотрена функция LambdaRank [12]. В данной моделе учитывались более, чем 50 признаков пользователей и элементов.

Была предложена архитектура двухуровневой рекомендательной системы, первым уровнем которого является модель факторизационных машин. Модель первого уровня для каждого пользователя $u \in U$ считает наиболее релевантных элементов-кандидатов. Вторым уровнем системы является контентая модель на основе градиентного бустинга, которая принимает для каждого пользователя элементы-кандидаты и выдает финальный список рекомендуемых элементов.

Все изученные алгоритмы и техники были программно реализованы и апробированны на дейтвительных данных. Двухуровневая модель улучшает качество рекомендаций на 53 процента по сравнению с одноуровневой рекомендательной системой.

Заключение

На сегодняшний день существуют множество различных алгоритмов рекомендаций, которые основываются на разных предположениях и используют различную информацию. Каждый алгоритм имеет свои достоинства и недостатки. В данной работе предпринимаются попытки объединить несколько различных подходов в одну гибридную рекомендательную систему. Данная система использует достоинства моделей коллаборативной фильтрации и контентных моделей.

Эксперименты показывают, что гибридная двухуровневая модель рекомендации показывает достаточно высокие результаты в сравнении с отдельными моделями. При этом вся гибридная архитектура не является ресурсоёмкой. Все эксперименты проводятся на реальных данных историй взаимодействий пользователей в одном онлайн-кинотеатре.

Список литературы

[1] Robin Burke. Hybrid Web Recommender Systems, pages 377–408. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[2] Koren, Yehuda; Bell, Robert; Volinsky, Chris (August 2009). "Matrix Factorization Techniques for Recommender Systems". Computer. 42 (8): 30–37.

[3] T. Hastie, R. Tibshirani, J. Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. Springer, 2016. 745 p.

[4] S. Rendle. Factorization machines. In Data Mining (ICDM), 2010 IEEE 10th International Conference on, pages 995–1000. IEEE, 2010.

[5] Maciej Kula. Metadata Embeddings for User and Item Cold-start Recommendations. arXiv preprint arXiv:1507.08439, 2015.

[6] J. Weston, S. Bengio, and N. Usunier. WSABIE: Scaling up to large vocabulary image annotation. In IJCAI, volume 11, pages 2764–2770, 2011

[7] J.H. Friedman. Greedy function approximation: A gradient boosting machine. Technical Report, IMS Reitz Lecture, Stanford, 1999; see also Annals of Statistics, 2001.

[8] K. Guolin, M. Qi, et al. LightGBM: A highly efficient gradient boosting decision tree. In NIPS, pages 3149–3157, 2017.

[9] Q. Wu, C.J.C. Burges, K. Svore and J. Gao. Adapting Boosting for Information Retrieval Measures. Journal of Information Retrieval, 2007.

[10] C.J.C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton and G. Hullender. Learning to Rank using Gradient Descent. Proceedings of the Twenty Second International Conference on Machine Learning, 2005.

[11] Tie-Yan Liu (2009), Learning to Rank for Information Retrieval, Foundations and Trends in Information Retrieval: Vol. 3: No 3, c. 225-331

[12] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. Learning, 11, pp. 23-581, 2010.

[13] Breitinger, Corinna; Gipp, Bela; Langer, Stefan (2015-07-26). Research-paper recommender systems: a literature survey. International Journal on Digital Libraries. 17 (4): 305–338.

[14] Акулич И. Л. Математическое программирование в примерах и задачах. — М.: Высшая школа, 1986. — С. 298-310.

[15] https://boosters.pro/championship/rekko_challenge/

[16] https://github.com/xaphoon/rekko_challenge

Приложение

Здесь будут приведены некоторые функции. Полная программная реализация находится в [16].

Функция подготовки данных для модели LightFM :

```
def get_dataset(data, ratings):
    def rank_fn(
        xs: np.ndarray, *,
        gammas: np.ndarray,
        gamma_default: float,
        threshold: float,
        k: int
    ) -> np.ndarray:
        assert np.all(gammas > 0)
        assert gamma_default > 0
        assert threshold > 0
        assert k > 0

        sign = np.sign(xs)
        xs = np.abs(xs)
        gammas = gammas.copy()
        gammas[xs >= threshold] = gamma_default

        return sign * np.floor(k * (xs / threshold) ** gammas)

    ratings_stats = ratings.groupby('user_uid').agg(
        {'rating': ['count', 'mean']}
        )
    ratings_stats.reset_index(drop=False, inplace=True)
    pandas_flatten_columns(ratings_stats, inplace=True)

    ratings = ratings.merge(
        ratings_stats,
        on='user_uid',
        how='left',
```

```
    copy=False
)

ratings.loc[
    ratings.loc[:, 'rating_count'] < 5,
    'rating_mean'
] = 6.5

ratings.loc[:, 'watched_ratio'] = np.greater_equal(
    ratings.loc[:, 'rating'].values,
    ratings.loc[:, 'rating_mean'].values
).astype(np.float32) * 2.0 - 1.0

ratings.drop(
    ['rating_mean', 'rating_count', 'rating'],
    axis=1,
    inplace=True
)
ratings.loc[:, 'consumption_mode'] = 'S'

purchase_mask = data['consumption_mode'] == 'P'
rent_mask = data['consumption_mode'] == 'R'

watched_ratio_truncated = np.clip(
    data.loc[:, 'watched_ratio'].values,
    -3, 3
)

gammas = 1.5 * np.ones_like(watched_ratio_truncated)
gammas[rent_mask] = 1.0
gammas[purchase_mask] = 0.7

data.loc[:, 'rank'] = rank_fn(
    watched_ratio_truncated,
    gammas=gammas,
```

```
    gamma_default=0.7,
    threshold=1.5,
    k=5
)

data = clean_by_interactions_iteratively(
    data,
    min_user_interactions=3,
    min_item_interactions=15,
    max_iter=5
)

abs_rank_vec = np.abs(data['rank'].values)
data.loc[:, 'weight'] = np.divide(
    abs_rank_vec,
    abs_rank_vec.max()
).astype(np.float32)

dataset = data.loc[:,
    ['user_uid','element_uid','rank','weight']
]

dataset['user_uid'] = dataset['user_uid']
dataset['element_uid'] = dataset['element_uid']


user2cat = dict(zip(
    dataset['user_uid'].cat.categories,
    range(len(dataset['user_uid'].cat.categories))
))

item2cat = dict(zip(
    dataset['element_uid'].cat.categories,
    range(len(dataset['element_uid'].cat.categories))
))
```

```
    interaction_dataset = sparse.coo_matrix(
        (
            np.clip(dataset['rank'], 0, 1),
            (
                dataset['user_uid'].cat.codes.copy(),
                dataset['element_uid'].cat.codes.copy(),
            )
        ),
        shape=(
            len(dataset['user_uid'].cat.categories),
            len(dataset['element_uid'].cat.categories)
        ),
        dtype=np.float64
    )

    train_dataset, test_dataset, train_weight, test_weight =
    dataset_split(
        interaction_dataset,
        dataset['weight'],
        test_size=.15
    )

    return train_dataset, test_dataset, train_weight,
           test_weight, user2cat, item2cat
```

Функция отбора кандидатов для модели второго уровня:

```
def recommend_for_user(
    predictions, *,
    filtered_elements,
    elements,
    elements_count=None
):
    # compute candidate elements for user
```

```
    predicted_elements = []

    sorted_indices = np.argsort(-predictions)
    sorted_elements = elements[sorted_indices]

    n_items = 0

    for element in map(np.asscalar, sorted_elements):
        if element in filtered_elements:
            continue

        predicted_elements.append(element)
        n_items += 1

        if elements_count is not None and
            n_items >= elements_count:
            break

    return predicted_elements

def make_candidates(
    model,
    user_filtered_elements,
    max_candidates,
    users,
    acceptable_elements=None,
    cnd_dict=None,
    cnd_multiplier=None,
):

    if cnd_dict is not None:
        cnd_size = {k: len(v) for k, v in cnd_dict.items()}
        cnd_users = set(cnd_dict.keys())

    results = {}
```

```
batch_size = 50000
users_for_prediction = list((
    set(model.domain.users.categories) & users
))
items_for_prediction = np.array(model.domain.items.categories)

if acceptable_elements is not None:
    items_for_prediction = np.array(
        [item for item in items_for_prediction if
        item in acceptable_elements]
    )

n_users = len(users_for_prediction)
n_batches = n_users // batch_size +
    (0 if n_users % batch_size == 0 else 1)


for batch_idx in range(1, n_batches + 1):
    begin = batch_size * (batch_idx - 1)
    end = batch_size * batch_idx
    batch_users = users_for_prediction[begin:end]

    predictions = model.predict(
        users=batch_users,
        items=items_for_prediction
    )

    for idx, user in tqdm(enumerate(batch_users)):
        if cnd_multiplier is not None:
            elements_count = min(
                cnd_size[user] * cnd_multiplier,
                max_candidates
            )
        else:
```

```
                elements_count = max_candidates

            candidates = recommend_for_user(
                predictions=predictions[idx],
                elements_count=elements_count,
                elements=items_for_prediction,
                filtered_elements=user_filtered_elements[user]
            )

            results[user] = {}
            results[user]['items'] = candidates
            if cnd_dict is not None:
                consumed_elements = cnd_dict[user]
                results[user]['target'] = [int(item in
                    consumed_elements) for item in candidates]

    return results
```