Содержание

Аннотация

Данная курсовая работа представляет собой реализацию рекомендательной системы и API для сайта курсов дополнительного образования и повышения квалификации НИУ ВШЭ. Рекомендательная система основана на гибридной модели, объединяющей два популярных подхода: ALS (Alternating Least Squares) и BM25.

В работе использован язык программирования Python для разработки системы. Контентная и коллаборативная модели были взяты из открытой библиотеки implicit, а API реализовано посредством библиотеки FastApi.

ALS является коллаборативным методом фильтрации, который анализирует исторические данные о предпочтениях пользователей и использует неявную обратную связь для дальнейшей рекомендации курсов. Однако, чтобы улучшить точность и релевантность рекомендаций, в системе также использован алгоритм BM25. Это алгоритм информационного поиска, который мы используем для оценки схожести курсов,основываясь на взаимодействиях пользователей.

Гибридная модель ALS + BM25 объединяет преимущества коллаборативной фильтрации и контентного подхода, обеспечивая более точные и персонализированные рекомендации.

Ключевые слова

Рекомендательная система, ALS, BM25, гибридная модель, API

1 Введение

1.1 Описание предметной области

Рекомендательные системы в наше время используются повсеместно, для улучшения опыта пользователя от использования продукта. В нашем случае мы хотим упростить поиск для посетителей сайта, а также добавить опции для продвижения менее популярных курсов.

Особенностью данной предметной области является небольшое количество объектов рекомендации. Они исчисляются в сотнях, в то время как на сайтах рекомендаций фильмов и музыки объекты исчисляются тысячами, на сайтах рекомендаций картинок доходит до миллионов. Так же стоит отметить, что сами пользователи часто хотят просмотреть курсы лишь по своей специфике работы, что уменьшает возможную вариативность рекомендации к минимуму. Это посодействовало в принятии некоторых решений в работе, о чем будет далее.

1.2 Постановка задачи

Курсы на платформе представлены как страницы с некоторой информацией: название, описание, длительность курса, тип обучения и другие данные. Пользователи взаимодействуют со страницей посредством переходов, кликов, отправки форм и т.д. Наша задача: имея данные о курсах и неявную обратную связь от пользователей, предоставить им рекомендацию.

1.3 Обоснование значимости системы

Целью нашего проекта стала рекомендательная система на сайте каталога программ дополнительного образования НИУ ВШЭ. Мы считаем, что она будет очень полезной для улучшения пользовательского опыта и увеличения продаж. Вот наши аргументы, почему такая система должна быть полезной:

Персонализированные рекомендации: анализируя предпочтения и поведение пользователей на сайте, мы можем предлагать им курсы, которые лучше всего соответствуют интересам и потребностям каждого конкретного пользователя.

Улучшения пользовательского опыта: система может упростить процесс поиска курсов для пользователей. Вместо того чтобы искать вручную, пользователи могут просматривать рекомендации и быстро находить интересующие их варианты.

Увеличение конверсии: когда пользователи видят релевантные и привлекательные рекомендации, они склонны совершать больше покупок. Это может значительно повысить

конверсию на сайте и увеличить доходы.

Повышение удержания клиентов: если пользователи получают рекомендации, которые соответствуют их интересам, они склонны задерживаться на сайте и возвращаться снова в будущем. Рекомендации помогают создать более глубокую связь с пользователями, повышая их удовлетворенность и лояльность к платформе.

Кросс-продажи и увеличение среднего чека: рекомендательная система будет предлагать пользователю дополнительные курсы, которые могут быть интересны ему на основе предыдущих выборов. Например, если пользователь приобрел курс по менеджменту, система может рекомендовать ему курсы, связанные с бизнесом или финансами. Таким образом, смежные по тематике курсы будут пополнять друг друга новыми пользователями, что увеличит кросс-продажи. В целом, рекомендательная система помогает улучшить пользовательский опыт и предоставляет значимые преимущества для бизнеса, помогая повысить эффективность продаж.

1.4 Структура работы и разделение задач в команде

Так как наша команда изначально не знала ничего про рекомендательные системы, то происходило долгое и кропотливое изучение материалов по этой теме (2) и (**??**). Поняв в каком направлении двигаться мы приступили к предобработке данных (3). После мы выбрали библиотеку с которой хотели работать для создания модели рекомендательной системы, однако позже нам пришлось этот выбор изменить. Об этом подробнее тут (4.1), (4.2).

Затем работа нашей команды разделилась:

Мария занялась детальным изучением и применением конкретных моделей (4).

Андрей занялся реализацией системы и написанием скриптов для обучения (5), (6).

2 Изучение поставленной задачи

2.1 Термины и определения

Объект рекомендации — рекомендуемый предмет, например, в нашем случае это курс, который может заинтересовать пользователя. Объект описывается своими характеристиками, в нашем случае: название, описание, длительность, факультет-организатор, формат обучения, тип программы.

Пользователь — человек, для которого генерируются рекомендации. Для того чтобы строить рекомендации нам нужно считывать и понимать поведение и предпочтения пользователей из их обратной связи.

Явная обратная связь — тип обратной связи, обеспечиваемый посредством прямых действий, таких как выставление оценок объектам, предоставление отзывов или заполнение опросов. Этот тип обратной связи является четким и прямым, но он часто бывает необъективным и ненадежным, поскольку пользователи не всегда честно говорят о своих предпочтениях, например, при оценке фильмов пользователи редко используют полный диапазон оценок, в основном ставят 7-8 из 10.

Неявная обратная связь — тип обратной связи, генерируемый на основе действий пользователя, например, нажатия на объекты, добавление объектов в закладки или обычный их просмотр. Этот тип обратной связи не требует от пользователя прямой оценки объектов, но он более надежный, поскольку отражает реальное поведение пользователя.

Проблема холодного старта в рекомендательных системах — это проблема, которая возникает, когда система пытается составить рекомендации для нового пользователя или объекта, по которому нет известных данных. Отсутствие данных может привести к неправильным рекомендациям.

API (Application Programming Interface) — это набор программных инструкций и протоколов, которые определяют способы того, как разные компоненты программного обеспечения должны взаимодействовать друг с другом, обмениваться данными и выполнять определенные функции.

CPU-bound — термин, означающий, что скорость выполнения процесса ограничена скоростью процессора. Например, перемножение матриц.

I/O-bound — термин, означающий, что скорость выполнения процесса ограничена ожиданиями ввода/вывода. Например задача чтения/записи на диск или в БД.

2.2 Общие подходы

Наиболее распространенными методами построения рекомендательных систем являются:

Collaborative Filtering: дает рекомендации на основе прошлого поведения и предпочтений пользователей, похожих на текущего пользователя.

Content-Based Filtering: дает рекомендации на основе характеристик объектов, которые нравились пользователю в прошлом. Метод генерирует рекомендации путем сопоставления характеристик, которые нравятся пользователю, с характеристиками других объектов.

Hybrid Recommendation Systems: сочетает в себе два предыдущих метода для создания более точных рекомендаций.

Deep Learning: использует нейронные сети для построения рекомендаций путем изучения закономерностей и связей в данных.

Гибридные модели часто считаются лучшим выбором для рекомендательных систем, поскольку они объединяют сильные стороны нескольких методов для получения более точных и разнообразных рекомендаций. Объединяя информацию из различных источников, гибридные системы могут преодолеть ограничения отдельных алгоритмов и обеспечить лучшие результаты.

Также в крупных продуктовых решениях не редко используется глубокое обучение, однако данный метод требует больших вычислительных ресурсов и не подходит к нашей задаче.

2.3 Обзор и сравнительный анализ подходов и аналогов

В начале работы над задачей мы изучили различные подходы к построению рекомендаций, конкретные решения и модели, отражающие каждый из подходов [2], [1] .

Content-based подход:

1 Построение рекомендаций по признакам объектов, которое использует косинусное сходство с объектами, с которыми пользователь уже взаимодействовал

2 Улучшение и ускорение этого решения, которое использует для рекомендаций только K ближайших соседей для каждого из объектов.

Collaborative подход:

1 **Sparse Linear method** — принцип данной модели заключается в том, что мы хотим не

явно посчитать матрицу сходства объектов, а подобрать ее так, чтобы минимизировать функцию потерь.

2 **ALS** — модель матричной факторизации, которая подбирает разложение матрицы взаимодействий на 2 матрицы так, чтобы минимизировать функцию потерь между реальными взаимодействиями пользователя и предмета и произведением подобранных латентных векторов пользователя и предмета. Алгоритм ALS чередует оптимизацию векторов пользователя и предмета, сохраняя один постоянным при обновлении другого.

3 **FunkSVD** — модель матричной факторизации, базирующаяся на ALS, цель остается такой же — найти приближенное разложение к матрице взаимодействий. В данном подходе сначала оптимизируется разложение с 1-компонентными латентными векторами (методом ALS), после сходимости добавляем еще один латентный признак и оптимизируем его и так далее.

4 **Asymmetric SVD** — модель матричной факторизации, которая решает проблему рекомендаций для пользователя, на котором не обучалась модель, мы вычисляем латентные вектора для пользователя в виде произведения вектора взаимодействий этого пользователя на матрицу из латентных векторов для объектов (Z). Для предсказания «рейтинга» взаимодействия перемножаются матрицы — RZY, где матрицы Z, Y подбираются.

Hybrid модели:

1 Комбинация content-based и collaborative моделей (получение итоговых рекомендаций, например, линейной комбинацией рейтингов моделей или слиянием уже готовых списков).

2 Матричная факторизация с дополнительными характеристиками, например, характеристиками объектов или признаками пользователей.

3 Модели, использующие контекст взаимодействия и матричную факторизацию.

Плюсы content-based моделей:

- Модель не требует данных о других пользователях, что позволяет легко масштабировать ее для множества пользователей.
- Она может предлагать рекомендации, учитывая индивидуальные интересы пользователя, в том числе элементы, интересующие небольшое количество других пользователей.

Недостатки:

- Модель может предложить рекомендации только по имеющимся предпочтениям пользователя. Таким образом, модель ограничена в своих возможностях по увеличению интересов пользователя.

Плюсы collaborative filtering и, в частности, матричной факторизации:

- Модель может предоставить возможность пользователям открыть для себя новые интересы. Хотя сама система может не знать о заинтересованности пользователя в конкретной области, модель все равно может рекомендовать ее, основываясь на том, что другие пользователи с похожими интересами заинтересованы в этом.
- Легко реализуется и не требует много информации о рекомендуемых предметах.

Недостатки:

- Проблема «холодного старта»: для многих моделей затруднительно составить рекомендации для новых пользователей или предметов без достаточного количества данных о взаимодействии.
- Снижение производительности при масштабируемости
- В связи с тем, что этот подход основывается на исторических взаимодействиях, модель не будет рекомендовать новые или малоизвестные продукты.

Мы остановили свой выбор на гибридных моделях, так как они помогают компенсировать недостатки отдельно взятых моделей, сделать рекомендации лучше. И в первом приближении мы решли работать с моделью матричной факторизации с дополнительными характеристиками с помощью библиотеки LightFM. Позже, поняв, что эта библиотека нам не подходит мы стали использовать Implicit.

3 Предобработка и анализ

3.1 Неявная обратная связь

Выполнил: Курдун Андрей

Для нашей работы нам был предоставлен набор данных в сыром виде, который имел около трех миллионов строк с колонками: пользователь, дата события, ссылка и тип действия.

Для начала я занялся тем, что почистил данные от не валидных ссылок и удалил все лишнее, что присутствовало в адресах страниц (чаще всего это были различные ключи и значения рекламных переходов). Отсоединил якоря и положил их в отдельную колонку. Разбил данные на кластеры по пользователям и их сессиям.

По этим данным нужно построить матрицу взаимодействий, где строки — это пользователи, колонки — это курсы и на их пересечениях стоит коэффициент, отражающий степень взаимодействия — число, полученное суммой выставленных весов за те или иные действия пользователя на странице курса. Например, отсутствие взаимодействий имеет коэффициент 0, за то, что пользователь просто посещал страницу когда-либо он получает небольшую прибавку, за то, что совершал на этой странице какие-либо действия получал еще немного.

Я выделил 5 основных типов взаимодействия пользователя со страницей:

1. Обычное посещение без взаимодействия с самой страницей курса

2. Какие-либо действия на странице

3. Переход по якорям

4. Кол-во сессий пользователя на этой странице

5. Отправка форм, всего их там 2 типа: записаться на курс, и получение обратной связи. Этот вид является самым стоящим, так как предполагает прямую заинтересованность пользователя

Далее я построил функцию для выставления коэффициента взаимодействия. Выглядит она так:

$$coefficient(user_i, item_j) = 0.08 \cdot has\_interaction + has\_submit\_form\_event +$$

$$+ min(0.2,\ 0.4 \cdot \log_{100000}(1 + \cdot\#interacions(user_i, item_j))) +$$

$$+ min(0.4,\ \log_{1000}(1 + \#anchors\_interactions(user_i, item_j)))$$

Где:

$has\_interaction$ 1 или 0 в зависимости от того есть ли взаимодействия

$has\_submit\_form\_event$ 1 или 0 в зависимости от того есть ли событие отправки формы

$\#interacions(user_i, item_j)$ кол-во взаимодействий пользователя и курса

$\#anchors\_interactions(user_i, item_j)$ кол-во взаимодействий пользователя и курса, где пользователь переходил по якорю

3.2 Описания курсов

Выполнила: Петухова Мария

Для построения более точных рекомендаций мы решили использовать характеристики курсов для обучения, я выполнила предобработку этого датасета, поменяла типы столбцов, проверила данные на корректность, исследовала столбцы на наличие пользы и информативности для построения рекомендаций, удалила ненужные столбцы, заполнила пропуски

Применила кодирование one-hot-encoding для таких категориальных признаков, как тип, формат обучения, и название факультета-организатора. Для кодирования самых информативных текстовых признаков — описание и название курса я решила использовать word2vec, но для начала надо было сделать предобработку и очистку текста [6]: приведение к нижнему регистру, удаление не буквенных символов, знаков препинания; предложения были разбиты на токены, удалены стоп-слова (предлоги, частицы и так далее), для этого была использована библиотека nltk. Также была проведена лемматизация (приведение слов в начальную форму), для этого была использована библиотека pymorphy2. Вначале планировалось обучить модель на обработанных текстах, используя библиотеку gensim, и получить нужные векторы, однако после анализа полученных векторов оказалось, что количество предложений для построения зависимостей модели было недостаточно, поэтому было принято решение использовать готовую предобученную модель для русского языка, которая ставит в соответствие слову вектор из 300 координат. Для использования этой модели нужно было добавить к каждому слову часть речи. Далее после получения вектора для каждого слова из предложения я использовала метод TF-IDF для получения вектора для всего текста описания/названия. Также для того чтобы получить более устойчивую модель и лучшие результаты, я применила нормализацию признаков.

3.3 Тестовая и обучающие выборки

Изначально мы отделили 20% взаимодействий (количество пользователей осталось тем же) для тестовой выборки с помощью *implicit.evaluation.train_test_split*. Оставшуюся обучающую выборку мы поделили построчно с помощью *sklearn.model_selection.train_test_split*, оставив 50% пользователей, чтобы можно было перебрать больше гиперпараметров и подбор происходил быстрее. Подбор параметров мы проводили на укороченной обучающей матрице, а при итоговом тестировании использовали полные обучающую и тестовую выборки.

4 Используемые методы для решения задачи

Выполнила: Петухова Мария

Принятые решения упомянутые в главах (4.1) и (4.2) обсуждались коллективно

4.1 Работа с библиотекой LightFM

Изначально мы начали работу используя библиотеку LightFM. Изучили то, как работают модели в данной библиотеке [5]. Однако уже в ходе работы было обнаружено некоторое количество проблем:

- В реализации LightFM каждому признаку курса сопоставляется латентный вектор, длина которого является гиперпараментром. Если длина будет недостаточно большой, то при суммировании этих векторов мы будем терять часть важной информации, так как у нас нет достаточного объема для ее сохранения. При высоком значении длины у нас получается еще одная большая матрица, которую мы теперь используем везде при обучении и рекомендациях. Так как на этом этапе мы уже представляли какое у нас количество признаков у курсов, то было очевидно, что использование данной библиотеки утяжелит все вычисления, в силу выше сказанной особенности.

- Скорость вычислений. LightFM тем эффективнее распараллеливает вычисления, чем больше взаимодействий у пользователя с объектами, в силу специфики отрасли нашего продукта, наши пользователи в принципе посещают не много курсов, поэтому такой подход к вычислениям нам не подошел.

- Гибридные модели предоставлены как готовое решение и они сложно настраеваемы. Например библиотека очень хороша для решения проблемы холодного старта, так как она может основывать свои рекомендации на дополнительной информации о пользователе: пол, возраст, образование, род деятельности и т.д. Однако в нашем случае у пользователей нет авторизации и возможности заполнить какую-либо информацию о себе. Получается у нас нет признаков пользователей, чтобы воспользоваться этой функциональностью.

- Используя контентную и коллаборативную модели раздельно будет значительно проще интерпретировать результаты, а также у нас появится большая гибкость и вариативность в настройке.

4.2 Работа с библиотекой Implicit

После некоторого исследования возможных вариантов наш выбор пал на библиотеку implicit [13], которая работает именно с неявной обратной связью.В данной библиотеке представлены модели матричной факторизации, то есть такие модели, которые строят разложение разреженной матрицы взаимодействий пользователей с курсами на две матрицы, которые хранят в себе латентные представления пользователей и курсов соответственно. Представленные модели : Alternating Least Squares, Bayesian Personalized Ranking и Logistic Matrix Factorization. Также данная библиотека предоставляет контентные модели, такие как Cosine Similarity, TF-IDF и BM25.

4.3 Выбор коллаборативной модели

Для каждой из трех коллаборативных моделей мы подбирали наилучшие гиперпараметры (для случая бинарной интеракции и для случая весов) с помощью кросс-валидации. Для этого я написала отдельные функции. Метрика, выбранная для определения лучших гиперпараметров - precision@k, так как она легко интерпретируема, ведь является отношением количества релевантных курсов в рекомендации из первых k курсов к min(k,#user interactions). Так как пользователи обычно не смотрят много рекомендаций, да и впринципе посещают мало страниц курсов, то мы выбрали k = 7. Также для более быстрого подбора гиперпараметров, матрица взаимодействий была обрезана по пользователям. Лучшей коллаборативной моделью стала модель ALS. Модель BPR дала немного хуже результат и Logistic еще хуже.

Таблица 4.1: Precision@7 у коллаборативных моделей с подобранными гиперпараметрами

	ALS	BPR	Logistic
с весами	0.409	–	0.154
без весов	0.432	0.397	0.195

Таблица 4.2: Наилучшие гиперпараметры у коллаборативных моделей

	factors	regularization	iterations	learning rate	neg prop
ALS с весами	190	2.0	100	–	–
ALS без весов	160	7.5	100	–	–
BPR без весов	60	0.03	100	0.08	–
Logistic с весами	30	2	200	4.0	40
Logistic без весов	40	1.5	200	0.5	34

4.3.1 ALS

Изученные статьи по этой теме: [8], [9].

Это модель матричной факторизации, которая представляет каждого пользователя и каждый курс как вектор длины f, характеризующий их через латентные признаки. Для того чтобы подбирать эти векторы, вводится степень "важности"для каждого взаимодействия, равная $c_{ui} = 1 + \alpha r_{ui}$, где r_{ui} - это или бинарный признак взаимодействия, или построенные по определенным правилам коэффициенты.

Функция потерь:

$$L = \sum_{u,i} c_{ui}(I(user\ u\ watched\ course\ i) - < x_u, y_i >)^2 + \lambda(\sum_u ||x_u||^2 + \sum_i ||y_i||^2)$$

где x_u, y_i - те самые векторы латентных признаков.

Далее мы по очереди фиксируем векторы пользователей/курсов, оптимизируя оставшиеся. Получается тогда, что точку минимума можно найти аналитически, а именно:

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

Y - матрица курсов и их латентных признаков

C_u - диагональная матрица со степенями важности взаимодействий

p_u - вектор с элементами $0/1$, которые показывают наличие взаимодействия

Аналогичное выражение можно записать и для векторов курсов

Учитывая,что обычно количество латентных признаков не более 200-300, то самая сложная операция в данном вычислении - это вычисление $Y^T C_u Y$, но, учитывая природу наших данных, а именно то что из множества курсов пользователь взаимодействует лишь с малым количеством из них, можно посчитать это умножение быстрее: $Y^T Y + Y^T (C_u - I) Y$, так как матрица $(C_u - I)$ преимущественно нулевая, а $Y^T Y$ маленькая.

4.3.2 BPR

Изученная статья по этой теме: [7].

В данной модели мы хотим максимизировать функцию правдоподобия

$$p(params\ of\ factorization | correct\ order)$$

Воспользуемся формулой Байеса:

$$p(params|correct\ order) = \frac{p(correct\ order|params)p(params)}{p(correct\ order)}$$

Будем оптимизировать то, что зависит от векторов в факторизации. Воспользуемся независимостью предпочтений пользователей, а также будем считать,что порядок какой-то конкретной пары независим с порядком других пар. Порядок в нашем случае - это стремление построить модель так, чтобы элементы с которыми пользователь взаимодействовал, шли раньше непросмотренных. Тогда мы хотим максимизировать

$$\Pi_{u,i,j\in\{(user,viewed,not\ viewed)\}} p(i > j|params) \cdot p(params)$$

где введем

$$p(i > j|params) = \sigma(\hat{x_{ui}} - \hat{x_{uj}})$$

где

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

В качестве априорного распределения будем считать что векторы латентных представлений распределены нормально с 0 матожиданием и ковариационной матрицей λI, где λ - параметр регуляризации.

Тогда мы хотим максимизировать

$$\sum_{u,i,j\in\{(user,viewed,not\ viewed)\}} \ln \sigma(\hat{x_{ui}} - \hat{x_{uj}}) - \lambda||params||^2$$

Далее мы оптимизируем латентные вектора используя градиентный спуск.

В нашем случае у нас есть дополнительная информация в виде степени взаимодействия с каждым из курсов, однако эта модель не использует эти коэффициенты, и это еще одна причина почему мы не выбрали ее.

4.3.3 Logistic MF

Изученная статья по этой теме: [3].

В данной модели мы опять стремимся максимизировать функцию правдоподобия (логарифм правдоподобия) и снова вводим степень важности $c_{ui} = 1 + \alpha r_{ui}$.Также мы воспользуемся снова формулой Байеса и выберем в качестве приора для распределения латентных векторов

- стандартное нормальное распределение с ковариационной матрицей λI.

$$L(observations|params) + \ln(P(params)) = \ln(\Pi_{u,i} p(user_u\ likes\ course_i|params)^{c_{ui}-1} \cdot$$

$$(1 - p(user_u\ likes\ course_i|params))) + \ln(P(params))$$

$$p(user_u\ likes\ course_i|params) = \frac{e^{<x_u,y_i>}}{1 + e^{<x_u,y_i>}}$$

Получается, мы стремимся максимизировать

$$\sum_{u,i}(c_{ui} - 1)(< x_u, y_i >) - c_{ui}\log(1 + e^{<x_u,y_i>}) - \lambda||params||^2$$

Для нахождения векторов максимизирующих эту функцию, часто используют алгоритм, подбирающий длину каждого шага градиентного спуска на основе прошлых градиентов, AdaGrad:

$$x_u^t = x_u^{t-1} + \frac{LearningRate \cdot g_u^{t-1}}{\sqrt{\sum_{p=1}^{t-1}(g_u^p)^2}}$$

где g_u^t - градиент на t шаге

4.4 Выбор контентной модели

Были подобраны наилучшие гиперпараметры для 4 моделей – контентная модель, основанная на вычислении косинусной схожести по интеракциям с весами, TF-IDF и BM25 основанные на матрице интеракций, а также, так как у нас были данные о самих курсах, (описание, длительность и так далее), то мы подобрали гиперпараметры для модели косинусной схожести по этим признакам. Подбор гиперпараметров был также по кросс-валидации с написанием соответствующих функций. Лучшей моделью стала BM25, чуть хуже TF-IDF и Cosine Similarity хуже всего.

4.4.1 BM25

Эта функция используется для оценки релевантности документов в поисковой выдаче на какой-то конкретный запрос, основана она на встречаемости каждого слова из запроса в конкретном документе, чем чаще встречается термин - тем лучше, также используется метрика IDF, которая вычисляет степень распространенности этого слова, тем самым мы уменьшаем показательность этого слова, если оно есть в большинстве документов, также

задействована длина самого документа. В нашем случае, множество документов - это множество пользователей, наш запрос - это курс, то есть мы считаем важность курса для пользователя, насколько этот курс отражает его предпочтения. Таким образом мы составляем матрицу весов размера $\#users \cdot \#courses$, где веса вычисляются по следующей формуле:

$$score_{ui} = IDF(i)\frac{TF(u,i)(K1+1)}{TF(u,i) + K1(1 - b + \frac{\#user_u\ interactions}{avg\ length})}$$

$$IDF(i) = \log \frac{\#users}{\#users, interacted\ with\ course_i}$$

$$TF(u,i) = \frac{I(user_u\ interacted\ with\ course_i)}{\#courses, user_u\ interacted\ with}$$

$$avg\ length = \frac{1}{\#users}\sum_{u=1}^{\#users} \#user_u\ interactions$$

Далее полученная матрица B используется для вычисления схожести курсов, путем вычисления матрицы $B^T B$. Score для рекомендаций формируется на основе данных о взаимодействии пользователя с K ближайшими соседями для этого курса.

$$\hat{r_{ui}} = \frac{\sum_{j \in KNN(course_i)} r_{ui} b_{ji}}{\sum_{j \in KNN(course_i)} b_{ji}}$$

Таблица 4.3: Precision@7 у контентных моделей с подобранными гиперпараметрами

	P@7
Cosine Similarity with courses features	0.281
Cosine Similarity on interactions	0.369
TF-IDF	0.429
BM25	0.437

Таблица 4.4: Наилучшие гиперпараметры у контентных моделей

	K	K1	B
Cosine Similarity with courses features	100	–	–
Cosine Similarity on interactions	131	–	–
TF-IDF	124	–	–
BM25	224	0.5	0.4

4.5 Метрики

Метрики, которые мы дополнительно анализировали при выборе наилучшей модели и гиперпараметров. Мы смотрели также на метрики ранжирования, так как нам важен поря-

док курсов в рекомендациях, ведь пользователи в основном смотрят на самые верхние курсы в выдаче.

$$Map@k = \frac{1}{N_{users}} \sum_{i=1}^{N_{users}} Ap@k(user_i)$$

Где $Ap@k(user_i)$ равно:

$$Ap@k(user_i) = \frac{1}{min(k, \#seen)} \sum_{i=1}^{k} \frac{(\#seen\ till\ i'th\ position) I(i'th\ seen)}{i} =$$

$$= \frac{1}{min(k, \#seen)} \sum_{i=1}^{k} P@i * I(i'th\ seen)$$

То есть для каждого пользователя мы считаем метрику, которая считает среднее precision на каждой из позиций релевантного элемента, таким образом, чем выше релевантные элементы в нашем списке рекомендаций, тем выше будет метрика, далее мы берем среднее по всем пользователям.

$$average\ nDCG@k = \frac{1}{N_{users}} \sum_{i=1}^{N_{users}} \frac{DCG@k(user_i)}{IDCG@k(user_i)}$$

Где $DCG@k(user_j)$ и $IDCG@k(user_j)$ равны:

$$DCG@k(user_j) = \sum_{i=1}^{k} \frac{I(i'th\ seen)}{\log(i+1)}$$

$$IDCG@k(user_j) = \sum_{i=1}^{min(k,\#seen)} \frac{1}{\log(i+1)}$$

То есть мы считаем для каждого пользователя сумму дробей с логарифмами в знаменателе для каждой позиции, где стоит релевантный курс, нормируем, а потом считаем среднее по всем пользователям

$$mean\ auc = \frac{1}{N_{users}} \sum_{i=1}^{N_{users}} auc@k(user_i)$$

$$auc@k(user_j) = \frac{\#pairs\ of\ viewed\ and\ not\ courses\ that\ are\ in\ correct\ order}{\#pairs\ of\ viewed\ and\ not\ courses}$$

То есть для каждого пользователя мы смотрим долю пар {просмотренных, непросмотренных} курсов, которые находятся в правильном порядке (просмотренные перед непросмот-

ренными)

4.6 Построение гибридной модели

Для того чтобы объединить успехи выбранных моделей и построить итоговую модель, которая бы рекомендовала курсы, основываясь на опыте других пользователей и основываясь на схожести с тем, что пользователь уже посмотрел, мы решили построить 2 гибридные модели с написанием классов для этих моделей и реализацией полного обучения, частичного обучения и рекомендаций : линейная модель с параметром альфа, которая строит рекомендации, вычисляя взвешенный score коллаборативной и контентной модели, и сортируя курсы по этому score, и Nfirst модель с параметром n, которая сливает 2 списка рекомендаций коллаборативной и контентной моделей (от коллаборативной модели n процентов рекомендаций, остальные от контентной), таким образом получившийся список чередует коллаборативную и контентную выдачу (без повторений). Перед подбором гиперпараметров для гибридных моделей, мы более точно подобрали гиперпараметры для контентной и коллаборативной моделей, используя библиотеку optuna и алгоритм TPE (Tree Parzen Estimator) [4]. Выбранный алгоритм использует идею байесовского подхода к подбору гиперпараметров: такой подход учитывает результаты прошлых оценок гиперпараметров и строит вероятностную модель p(score|hyperparams). Это помогает нам предсказывать и выделять перспективные области для поиска. Так как верна формула

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

то будем строить вероятностную модель

$$p(hyperparams|score) = \begin{cases} P1(hyperparams|score > score^*) = P1(hyperparams|bad\ score) \\ P2(hyperparams|score \leq score^*) = P2(hyperparams|good\ score) \end{cases}$$

От перспективных кандидатов логично ожидать высокую вероятность при условии хорошего score и низкую вероятность при условии плохого score. Поэтому хорошие кандидаты на гиперпараметры имеют высокий коэффициент $\frac{P2(hyperparams|good)}{P1(hyperparams|bad)}$

Соответственно, на этой идее и строится алгоритм: для начала гиперпараметры выбираются случайно, считаем на них score, и по нему их и разбиваем на 2 группы (в первой

- с самым лучшим score, во второй - оставшиеся). Далее для хорошей и плохой группы моделируем распределения P2 и P1 соответственно, строя ядерную оценку плотности, используя, например, нормальную яденую функцию. Далее семплим из смоделированного "хорошего"распределения P2 гиперпараметры, смотрим на их коэффициент $\frac{P2(hyperparams|good)}{P1(hyperparams|bad)}$, по нему выбираем самого перспективного, считаем его score, добавляем в множество уже опробованных гиперпараметров и вновь это множество разбиваем на 2 группы и проводим ту же процедуру достаточно много раз.

Таблица 4.5: Precision@7 у выбранных моделей с улучшенными гиперпараметрами

	previous P@7	new P@7
ALS with weights	0.409	0.418
ALS without weights	0.432	0.434
BM25	0.437	0.438

Таблица 4.6: Улучшенные гиперпараметры у выбранных моделей

	new hyperparams
ALS with weights	factors = 170, regularization = 1.765, alpha = 14.592, iterations = 137
ALS without weights	factors = 150, regularization = 8.98, alpha = 1.21, iterations = 50
BM25	K = 180, K1 = 0.513, B = 0.39

После этого мы подобрали гиперпараметры для гибридных моделей по кросс-валидации, выбранная метрика - precision@k с k = 10.

Таблица 4.7: Precision@10 у гибридных моделей

	P@10
Linear (ALS weighted)	0.5012
Linear	0.5011
Nfirst (ALS weighted)	0.499
Nfirst	0.499

Таблица 4.8: Гиперпараметры у гибридных моделей

	hyperparameter
Linear (ALS weighted)	0.6
Linear	0.65
Nfirst (ALS weighted)	0.1
Nfirst	0.1

Метрики, время обучения и рекомендаций не сильно отличались у моделей, однако как мы видим, модель Nfirst стремится к тому чтобы работала по факту только 1 контентная

модель,и лишь 10% (при малой выдаче это 1-2 курса) от коллаборативной модели, но все же мы хотели бы расширять границы пользователей и не зацикливаться на рекомендации одних и тех же курсов, максимально близких к просмотренным, а опираться на опыт других людей, поэтому мы все-таки выбрали Linear с весами (так как score немного лучше чем без весов и веса хранят в себе много информации о взаимодействиях).

4.7 Тестирование выбранной модели

Мы обучили итоговую гибридную модель на полной обучающей выборке и протестировали, полученные метрики:

Таблица 4.9: Итоговые метрики на тестовой выборке

	$K = 7$	$K = 10$
p@k	0.474	0.534
map@k	0.308	0.317
ndcg@k	0.370	0.390
auc@k	0.751	0.780

4.8 Рассмотрение крайних ситуаций

Если нужно составить рекомендацию пользователю, а модель не обучалась на нем и его не знает, но он уже провзаимодействовал с чем-то на сайте, то мы будем использовать метод recalculate user, который использует имеющийся вектор взаимодействий пользователя и находит вектор для пользователей такой что при имеющихся представлениях курсов достигается минимум функции потерь. Если про пользователя, на котором уже обучалась модель, появились новые взаимодействия, то мы будем рекомендовать те же курсы, но будем передавать в метод для рекомендаций полный список взаимодействий, чтобы в рекомендации не попали уже просмотренные курсы. Также раз в какое-то время будет происходить дообучение модели на новых взаимодействиях и пользователях. В случае если нам нужно порекомендовать что-то новому пользователю, у которого нет взаимодействий, мы будем выдавать просто список самых популярных курсов на платформе.

5 Реализация системы

Выполнил: Курдун Андрей

5.1 Выбор языка и библиотек

Необходимо было реализовать рекомендательную систему. Язык: python. Мы выбрали этот язык в силу того, что на нем есть уже некоторые готовые решения для рекомендательных систем, а также возможность упростить себе разработку api. К тому же хотелось бы чтобы код был не сильно сложным, чтобы в дальнейшем он мог обновляться и дополняться. Далее нужно было понять какие библиотеки мы хотим использовать. В конечном счете нам понадобились:

- **Numpy** для работы с массивами данных. Так же была возможность выбрать pandas или стандартные list из питона, но оба они давали недостаточную эффективность для решения runtime задач
- **Scipy.sparse** для работы с разреженными таблицами, которые нужны в дальнейшем для работы с моделями. Есть и другие библиотеки, которые предоставляют разреженные таблицы, но именно эту использует наша следующая библиотека для рекомендательных моделей, а также она является популярным и удобным решением, так как является расширением numpy
- **Implicit** предоставляет модели рекомендательных систем. В главах (4.1) и (4.2) более подробно описаны причины выбора данной библиотеки.
- **Fastapi** Веб-фреймворк для создания api на python [12]. Легок в конфигурации, крайне производительный, отлично подходит для развертывания моделей машинного обучения, так как направлен на создание RESTAPI поверх микросервисной среды. Также поддерживает из коробки асинхронные запросы.
- **Asyncio** Раз FastApi поддерживает асинхронность, то нужно использовать библиотеку для той самой асинхронности [10].
- **ProcessPoolExecutor из concurrent.futures** использовался для многопроцессорности, об этом и Asyncio подробнее будет тут (5.2)
- **Pytest** Для тестирования

- **Logging** Для логгирования

БД ClickHouse так как данные уже лежат в этой БД. Задача была в том, чтобы понять как из питона делать запросы к ней.

5.2 Принцип работы системы

Одной из самых значимых функции *load_updates_and_retrain* она отправляет запрос в бд и подгружает все необработанные системой взаимодействия пользователей, затем обрабатывает, фильтрует и сохраняет их в системе. Далее в зависимости от того, как давно мы последний раз переобучались, производим частичное/полное переобучение. Данную функцию мы запускаем при старте приложения, а также добавляем в планировщик при помощи asyncio, и вызываем раз в какое-то конфигурируемое время. Таким образом наша модель получает новые данные.

Далее возник вопрос как обучать модель так, чтобы это происходило эффективно с точки зрения CPU-bound задачи и при этом асинхронно, чтобы наше API могло отвечать пользователям в процессе обучения.

5.2.1 Выбор механизма конкурентных вычислений

В питоне существует 3 основных механизма организации конкурентных вычислений: многопоточность, многопроцессорность и асинхронность. Каждый из механизмов призван решать свою задачу и нам нужно будет выбрать какую-то их комбинацию.

Для начала обозначим, что конкурентные вычисления бывают двух типов: многозадачные и многопроцессорные.

Многозадачные механизмы позволяют в рамках одного процесса переключатся между задачами, что оптимизирует работу системы при взаимодействиями с I/O-bound задачами, так как вместо длительного ожидания, поток может пойти выполнять другое действие. Однако в Python, в силу существования GIL и его глобальной блокировки интерпретатора, эти механизмы не могут работать честно параллельно, т.е. каждый поток в механизме получает управление по очереди.

В то же время многопроцессорные механизмы решают задачи независимо, в связи с чем возможно исполнение на разных ядрах процессора, т.е. честно параллельно, что делает более эффективным решение CPU-bound задач, где вместо ожидания ответа от сети/операционной системы, мы большую часть времени активно что-то считаем.

Многозадачность также имеет разделение на два типа: вытесняющая и кооперативная. Вытесняющая подразумевает, что планированием задач занимается операционная система, в связи с чем переключение происходит в произвольный момент выполнения программы и требует траты лишних ресурсов на взаимодействие с ОС. В то же время кооперативная многозадачность требует от самой задачи указать места, где ей было бы удобно или оптимально отдать управление. Это эффективнее, так как не требует запросов к ОС, что существенно экономит ресурсы, а также дает гарантию потокобезопасности.

Про многопроцессорность можно сказать, что она заведомо имеет потокобезопасность, так как задачи выполняются в разных процессах, а значит не могут как-то помешать состояниям друг друга.

Так как у нас в системе есть и I/O-bound и CPU-bound задачи, то нам нужно комбинировать методы.

Библиотека Implicit умеет параллелизировать обучение сама, но она не умеет производить асинхронное выполнение, т.е. результат выполнения является для нас как бы I/O-bound задачей. Получается асинхронность в этом месте сделать не получится, нужно прибегать к многопоточности или многопроцессорности.

Когда я начал углубляться в подробности выбора того или иного метода, я пришел к мысли, что многопоточность заставит нас в любом случае создавать копию модели, так как чтобы во время обучения мы все еще могли давать рекомендацию, наша система должна поддерживать модель в консистентном состоянии. Вызов .fit функции нарушает эту консистентность. Но если мы уже создаем копию такой немаленькой структуры, то можно расширить функционал системы и выгружать модель на диск, что позволит при перезапуске не обучать ее заново.

Что такая опция дает относительно конкурентных вычислений: мы можем использовать многопроцессорность, подгружая модель и матрицу взаимодействий прямо из нового процесса.

Я посчитал решение с сохранением модели на диск и запуском из нового процесса более элегантным относительно интерфейса, так как не заставит нас заниматься вопросами потокобезопасности и разделит в разные процессы логику основной системы и обучения, что позитивно скажется на том, как ОС будет распределять ядра между ними.

5.2.2 Вмешивание курсов в рекомендацию

После того как мы получили рекомендацию наша система предлагает опциональную возможность: вмешать в результат курсы, из списков непопулярных или так называемых рекламируемых. Список непопулярных составляется системой автоматически, список рекламируемых настраивается.

Как это происходит: мы обучили Cosine Similarity на описаниях курсов. Затем используем функцию *similar_items* на рекомендованных курсах, ищем похожие среди списка рекламируемых и/или непопулярных. Затем добавляем результат в конец рекомедации.

Мы используем тут косинусную схожесть, так как хочется предоставить пользователю что-то действительно похожее по содержанию на то, что его интересует, а не по тому как другие пользователи взаимодействовали с курсом.

Сама идея вмешивания в том, чтобы продвигать непопулярные курсы или повышать продажи рекламиремых. Эта опция идет вне основного пайплайна модели и не участвует в исследованиях с метриками потому что она с большой долей вероятности ухудшает score, но, как уже было сказано, цель данного действия не в качестве.

В API рекомендации есть возможность выбрать какая часть рекомендации будет рекламируемой, и какая будет из непопулярных.

5.2.3 Конфигурация

Конфигурация разделена на sensitive и обычную для того, чтобы что-то стандартное можно было загрузить на гитхаб, а чувствительные остались только локально. Пример чувствительных данных в конфиге: данные для подключения к clickhouse.

Параметры для настройки системы представлены следующие: задержка между подгрузкой данными из БД, задержка для частичного/полного переобучения; различные настройки логгирования и путь для сохранения модели.

5.3 Описание API

/recomendation/

userid: str, fields_count: int, adv_perc: float, bottom_perc: float, as_link: bool

Возвращает рекомендацию. Здесь и далее *userid* – id пользователя, *fields_count* – максимальное кол-во полей, которые мы хотим увидеть в ответе, *as_link* – если False, то вернется id курса, если True, то вернется ссылка на курс.

/viewed/ userid: str, fields_count: int, as_link: bool

Если *fields_count* = 0 (по умолчанию), то возвращает все курсы, с которыми пользователь взаимодействовал, запроса в БД нет. Если не равно 0, то возвращает *fields_count* курсов, в порядке убывания таймстемпа взаимодействия. Для этого происходит запрос в БД.

/recomendation_and_viewed/
userid: str, rec_fields_count: int, viewed_fields_count: int, adv_perc: float, bottom_perc: float, as_link: bool

Комбинация двух прошлых запросов.

/user_coefficient/ userid: str, coursid: str

Скорее тестировочная возможность проверить кое-ф взаимодействия пользователя и курса.

/top_courses/ fields_count: int, as_link: bool
/bottom_courses/ fields_count: int, as_link: bool
/adv_courses/ as_link: bool

Список популярных, непопулярных и рекламируемых курсов.

/force_load_updates_and_full_retrain/

Форсированное переобучение модели

5.4 Обучение

Для обучения моделей мы написали скрипт который позволяет из консоли по выбранным аргументам тонко настраивать желаемую модель, а затем оставляет параметры лучшей из них в файле.

6 Развёртывание системы

Выполнил: Курдун Андрей

Также целью проекта было развернуть систему. Для этого мы использовали Docker [11].

Docker - это контейнизатор приложений который позволяет:

- Изолировать и автоматизировать разворачивание среды и запуск приложения. Часто хочется, чтобы сервис работал не на самой машине, где его запускают, а во внутренней песочнице, которую создали специально для него. Это позволяет гарантировать возможность установки нужных библиотек, а также позволяет минимизировать риски ошибок, которые могут прекратить работу хоста, так как происходит изолирование основной ОС от той, на которой работает сервис.

- Контроль ресурсов и возможность масштабировния. Так как мы сами создаем песочницу, то мы можем ограничивать ее в ресурсах, и наоборот, запускать параллельные копии которые в дальнейшем можно соединить при помощи балансера и увеличить производительность системы. Это уже отдельная тема, которую мы не затронули в нашей работе.

- Микросервисная архитектура. Сервис не обязательно интегрировать в другой буквально подключая его как библиотеку. Можно запускать все в отдельных контейнерах и работать с каждым сервисом независимо.

Мы собрали образ, который содержит код системы и поднимает необходимую для его запуска среду.

Список литературы

[1] Paolo Cremonesi. *Recommender Systems Course Notes.*

[2] Darel13712. *Рекомендательные системы: идеи, подходы, задачи.* URL: https://habr.com/ru/companies/jetinfosystems/articles/453792/ (дата обр. 29.05.2019).

[3] Christopher C. Johnson. "Logistic Matrix Factorization for Implicit Feedback Data". В: *web.stanford.edu* (2014).

[4] Will Koehrsen. "A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning". В: *Towards Data Science* (2018).

[5] Maciej Kula. "Metadata Embeddings for User and Item Cold-start Recommendations". В: *arXiv:1507.08439, version 1* (2015).

[6] Data Monsters. "Text Preprocessing in Python: Steps, Tools, and Examples". В: *Product AI* (2018).

[7] Steffen Rendle Christoph Freudenthaler Zeno Gantner Lars Schmidt-Thieme. "BPR: Bayesian Personalized Ranking from Implicit Feedback". В: *arXiv:1205.2618, version 1* (2009).

[8] Victor. "ALS Implicit Collaborative Filtering". В: *Rn Engineering* (2017).

[9] Yifan Hu Yehuda Koren Chris Volinsky. "Collaborative Filtering for Implicit Feedback Datasets". В: *yifanhu.net* (2008).

[10] *Документация asyncio.* URL: https://docs.python.org/3/library/asyncio.html.

[11] *Документация docker.* URL: https://docs.docker.com/.

[12] *Документация fastapi.* URL: https://fastapi.tiangolo.com/ru/.

[13] *Документация implicit.* URL: https://benfred.github.io/implicit/.