Федеральное государственное образовательное бюджетное учреждение
высшего профессионального образования
«Финансовый университет при Правительстве Российской Федерации»

Департамент анализа данных, принятия решения и финансовых технологий

Курсовая работа

по дисциплине **"Технологии анализа данных и машинное обучение"** на тему:

"Разработка и построение рекомендательной системы"

Вид исследуемых данных: Рейтинг и история просмотра пользователей аниме-сериалов

Выполнила:
студентка группы ПМ3-1
Баданина Н. Д.
Научный руководитель:
к.э.н., доцент
Гринева Наталья Владимировна

Москва 2020

Содержание

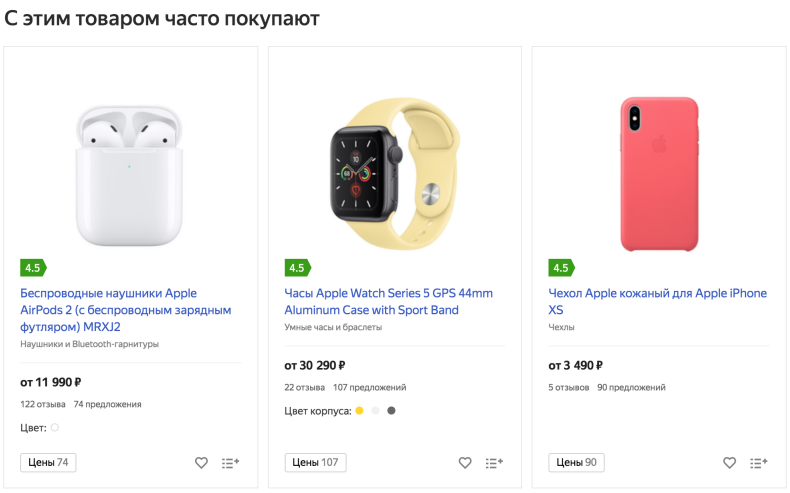
Введение

Долгое время маркетинг был традиционным. До появления современных технологий продавец в локальном магазине знал о своих покупателях гораздо больше, чем требовалось для продажи. Можно сказать, что предложения продавца покупателю были персонализированными или таргетинговыми.

С развитием современных технологий и экстенсивным ростом индустрии, большие бизнесы, получающие доход с продаж, уже не знают своих клиентов лично. Обычно о клиенте известные общая информация, например пол или регион проживания. На этом этапе появляется не персонализированная реклама: баннеры, вывески. Телевидение, несмотря на свой охват, тоже предлагает не таргетинговую рекламу. На основе данных об аудитории канала рекомендации будут неэффективные: слишком общие или неактуальные.

Проблема сбора данных решается с появление сети Интернет. Сейчас о пользователе можно узнать геопозицию, каким телефоном и браузером он пользуется, личные данные и увлечения из профилей социальных сетей. Особенностью сбора данных через интернет является то, что пользователь сам указывает начальную информацию о себе: ФИО, город, номер телефона. Остальные данные получают исходя из поведения клиента. Однако, и при этом подходе, рекомендации могут быть слабо персонализированными в силу специфичности данных. На этом моменте построения рекомендаций необходимо применять алгоритмы машинного обучения, которые, обработав массив данных о пользователе, могут сделать неочевидный вывод о том, какой товар или услугу можно продать клиенту.

Алгоритмы машинного обучения, которые предсказывают какой фильм/статью/товар/услугу купит пользователей называют рекомендательными системами. В основном такие системы используются для рекомендаций фильмов (Netflix, YouTube), товаров (Яндекс.Маркет, Ozon), музыки (Яндекс.Музыка) и статей (Яндекс.Дзен).



Например, так выглядят рекомендации к товару Iphone 11 Pro на

Яндекс.Маркет. Можно заметить недоработку: рекомендуют купить чехол к другой модели телефона.

Клиент получает рекомендацию, а сервис зарабатывает на предоставлении услуг. Услуги в данном контексте не обязательно значат продажи предлагаемого товара. Сервис имеет возможность зарабатывать на комиссионных, показе рекламы или увеличивать лояльность пользователей, которые потом будут покупать только в этом магазине. Интересно, что даже страницы в поиске сейчас персонализированы и разным пользователям по одному и тому же запросу могут быть предложены разные ссылки на сайты.

В данной работе рассмотрено построение рекомендательной системы для совета аниме-сериала пользователю. По оценкам McKinsey, 75% выручки онлайн кинотеатра Netflix приходится именно на рекомендованные фильмы/сериалы и процент этот, вероятно, будет расти. Рынок японских анимационных сериалов только развивается в России, поэтому система будет актуальная для многих сервисов, таких как Ivi, Okko и Кинопоиск, которые ещё не внедрили контент этого типа на свои платформы.

1 Теоретическая часть

1.1 Обзор методов и алгоритмов рекомендательных систем

Рекомендательные сервисы многообразны, так как их основные характеристики могут сильно отличаться. Любую рекомендательную систему можно описать с помощью таких характеристик, как

– предмет рекомендации: товары (Ozon, Amazon), статьи (Яндекс.Дзен), новости (Surfingbird), люди (Linkedin, LonelyPlanet), музыка (Last.fm, Pandora), плейлисты и так далее. Обобщая, рекомендовать можно что угодно.

– цель. Для чего рекомендуется: информирование, покупка, обучение.

– контекст. Чем пользователь в этот момент занимается: смотрит товары, читает статью, слушает музыку.

– источник рекомендации. Кто рекомендует: схожие по интересам пользователи, аудитория (средний рейтинг ресторана в TripAdvisor), экспертное сообщество (такой случай встречается, когда речь идет о сложном товаре, таком, как например, вино).

– глубина персонализации. Неперсональные рекомендации – пользователю рекомендуют те же товары, что и всем остальным. Такие предложения допускают таргетинг по времени или региону, но не учитывают личные предпочтения и историю клиента. Более лучший вариант – рекомендации используют данные из текущей сессии. Пользователь смотрел некоторые товары, ему предлагаются похожие. Персональные рекомендации используют всю доступную информацию о клиенте, в том числе историю его покупок.

– прозрачность. Клиенты больше доверяют рекомендации, если понимают, как почему именно она была показана. Так меньше риск встретить некачественные системы, продвигающие проплаченный товар или ставящие более дорогие товары выше в рейтинге. Кроме того, хорошая система сама должна уметь бороться с купленными отзывами и накрутками продавцов. Манипуляции бывают и непреднамеренными. Например, когда выходит новый блокбастер, первым делом на него идут фанаты, соответственно, первую пару месяцев рейтинг может быть сильно завышен.

– формат рекомендации. Это может быть всплывающее окно, лента внизу экрана, появляющийся в определенном разделе сайта список.

– алгоритмы. К наиболее классическим относятся алгоритмы Content-based (модели основанные на описании товара), Summary-based (неперсональные), Matrix Factorization (методы основанные на матричном разложении), Collaborative Filtering (коллаборативная фильтрация). Некоторые из них были реализованы в разделе практики.

1.2 Постановка задачи рекомендательной системы

Задача рекомендательной системы — найти для пользователя такие элементы из заданного множества, которые с высокой вероятностью ему понравятся. Например: товары, которые пользователь захочет купить; фильмы, которые пользователю понравятся; статьи, которые пользователь дочитает до конца.

Введем некоторые стандартные обозначения. Пользователи — множество U (users). В этой работе в множество U попадают пользователи сайта, поставившие какому-либо аниме-сериалу оценку. Новые пользователи рассматриваться не будут, так как возникает проблема "холодного старта", при которой у программы недостаточно информации для рекомендации. Подробно эта проблема будет рассмотрена в разделе "Проблематика".

Товары — множество I (items). В множестве I могут быть любые категории товаров: и фильмы, и музыка, и техника — зависит от задачи. В этой работе в качестве множества I выступает список аниме-сериалов. Так же будем считать, что для некоторой пары пользователь-товар $u \in U$ и $i \in I$ известна оценка r_{ui}, выражающая заинтересованность пользователя в этом товаре.

Заинтересованность может выражаться в разных единицах. В случае с музыкой, это может быть как много раз пользователь слушал данный трек, как много раз дослушал до конца, лайкнул ли он его. В случае со статьями, дочитал ли он текст до конца, сколько раз кликнул, скорость проматывания страницы. В данной работе заинтересованность принимает вид рейтинга, поставленного пользователем аниме-сериалу и находящегося в промежутке от 1 до 10.

Требуется по известным оценкам заинтересованности научиться находить для каждого пользователя u набор из k товаров $I(u)$, наиболее подходящих пользователю, то есть таких, для которых оценка r_{ui} окажется максимальной. Следовательно, в итоге, задача сводится к тому, что для объекта вида x_{ui} надо уметь предсказывать оценку r_{ui}. Как и в любой задаче машинного обучения, требуется уточнить три пункта: целевая переменная, признаки, функционал ошибки. Первый пункт был рассмотрен выше. Необходимо рассмотреть признаки, характеризующие пару пользователь-товар.

1.3 Алгоритмы рекомендаций

1.3.1 Статистические признаки

Рассмотрим простые типы факторов. Среди них могут быть: количество покупок и просмотров данного товара, количество покупок пользователя в данной категории, количество покупок пользователей.

Если товар или пользователь уже набрали достаточно статистики, то часто такие признаки оказываются самыми главными при принятии решения, поскольку уже содержат в себе достаточно информации о предпочтениях. Так, используя эти признаки, можно рекомендовать популярные товары в той категории, что нравится пользователю.

1.3.2 Коллаборативная фильтрация

Методы коллаборативной фильтрации строят рекомендации для пользователя на основе схожести между пользователями и товарами. Различают несколько подходов:

— Memory-based

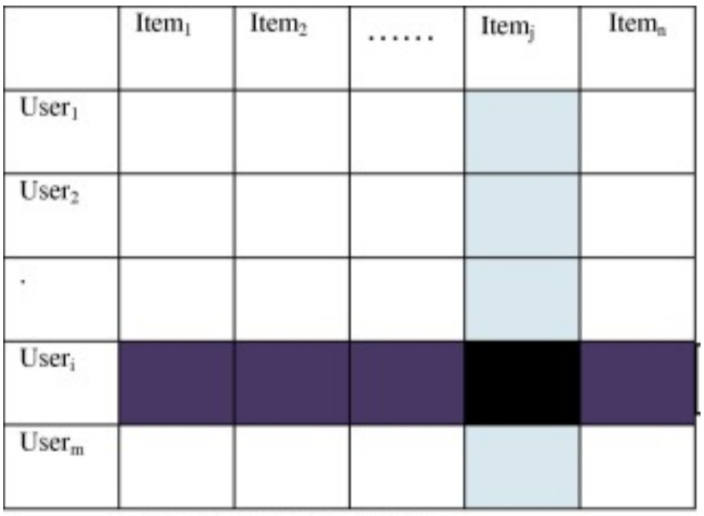
— Модели со скрытыми переменными

— Гибридный: объединение вышеперечисленных подходов.

Рассмотрим memory-based подход. В его основе лежит таблица R, размерности $N \times M$, где N — количество пользователей, а M— количество товаров. Это матрица, по одной из осей которой отложены все клиенты сервиса (Users), а по другой – объекты рекомендации (Items). На пересечении некоторых пар (user, item) данная матрица заполнена оценками (Ratings) – это известный показатель заинтересованности пользователя в данном товаре, выраженный по заданной шкале (например от 1 до 10). На пересечении строки u и столбца i хранится значение r_{ui}, если оно известно. Здесь стоит отметить, что, в основном, такая матрица будет состоять из нулей или пропусков, так как товаров много, а процент покупок, в отношении к количеству позиций, маленький. Вытекающим из этого замечания является вывод о том, что в программе необходимо хранить всю матрицу рейтингов, которая имеет большую размерность. Это не только в разы увеличивает время работы, но и усложняет визуализацию и процесс вычислений с использованием этой матрицы.

Пользователи обычно оценивают всего лишь малую часть товаров и задача рекомендательной системы – обобщить эту информацию и предсказать отношение клиента к другим товарам, про которые ничего не известно. Иными словами нужно заполнить все пустующие ячейки. Паттерны потребления у людей разные, следовательно система необязательно должна рекомендовать новые товары. Можно показывать повторяющиеся товары, например, для пополнения запаса. Исходя их этого принципа выделяют две группы товаров: повторяемые и неповторямые, например неповторямые – это книги или фильмы, которые

редко приобретают повторно. Выделяя повторяемые товары в категорию можно сократить размерность матрицы R.



Матрица R.

Некоторым пользователи покупают вещи только из их любимой категорий (conservative recommendations), а некоторые, напротив, лучше откликается на нестандартные товары или группы товаров (risky recommendations). Например, онлайн кинотеатр может рекомендовать пользователю только новые серии конкретного сериала, а может показывать новые шоу или новые жанры.

1.3.3 Baseline предсказание

Самое простое предсказание, которое можно сделать, имея таблицу R, посчитать среднее всех известных оценок и считать, что и остальным товарам пользователи будут ставить именно такие оценки:

$\mu = \frac{\sum_U \sum_I r_{ui}}{|Y|}$, где Y — множество известных оценок.

Но понятно, что такой подход имеет множество недостатков, один из них заключается в том, что не учитывается предвзятость пользователей и популярность товаров. Для того, чтобы учесть данные характеристики, найдем среднюю оценку для каждого товара и для каждого пользователя:

$\mu_i = \frac{\sum_U r_{ui}}{n_i}$

$\mu_u = \frac{\sum_I r_{ui}}{m_u}$, где n_i и m_u — количество оценок у товара i и пользователя u соответственно.

Посчитав сдвиги для пользователей и товаров относительно средней оценки, можно более точно вычислить, какую оценку пользователь u поставит товару i:

$b_i = \mu_i - \mu$

$b_u = \mu_u - \mu$
$r_{ui} = \mu + b_i + b_u$

Однако, при этом подходе никак не учитывается схожесть пользователей. Получается, что при подсчете средней оценки на неё влияли как пользователи со схожими вкусами, так и те, вкусы которых сильно разнятся. Поэтому необходимо модифицировать подсчет среднего: считать среднее для всех товаров для выбранного пользователя u_0, при этом влияние других пользователей будет зависеть от их схожести с выбранным пользователем $sim(u_0, u)$:

$$\mu = \frac{\sum_U sim(u_0,u) \sum_I r_{ui}}{|Y|}$$

Соответственно также можно модифицировать подсчет средней оценки для товаров.

1.3.4 Сравнение пользователей и товаров

Возникает вопрос: как сравнивать схожесть пользователей. Необходимо исходить из следующего предположения: два пользователя считаются похожими, если они ставят одним и тем же товарам одинаковые оценки. Рассмотрим двух пользователей u и v. Обозначим через I_{uv} множество товаров i, для которых известны оценки обоих пользователей:

$$I_{uv} = \{i \in I | \exists r_{ui} \& \exists r_{vi}\}$$

Тогда сходство между двумя данными пользователями можно вычислить с помощью корреляции Пирсона:

$$sim(u, v) = \frac{\sum_{i \in I_{ui}} (r_{ui} - \mu_u)(r_{vi} - \mu_v)}{\sum_{i \in I_{uv}} (r_{ui} - \mu_u)^2 \sum_{i \in I_{uv}} (r_{vi} - \mu_u)^2}$$

Основной минус расчета по корреляции Пирсона — корреляция может иметь большое значение случайно. Чтобы предотвратить завышение корреляции можно домножить корреляцию Пирсона на коэффициент, значение которого уменьшается с ростом числа оценок.

Чтобы вычислить сходство между товарами i и j также введем множество пользователей U_{ij}, для которых известны оценки для этих товаров.

$$U_{uj} = \{u \in U | \exists r_{ui} \& \exists r_{uj}\}$$

Тогда сходство двух товаров можно вычислить через корреляцию Пирсона:

$$sim(i, j) = \frac{\sum_{u \in U_{ij}} (r_{ui} - \mu_i)(r_{uj} - \mu_j)}{\sum_{u \in U_{ij}} (r_{ui} - \mu_i)^2 \sum_{u \in U_{ij}} (r_{ui} - \mu_j)^2}$$

Можно применять и другие способы вычисления схожестей. Например, вычислить норму разности между векторами или скалярное произведение векторов.

Существует три метода, использующих сходство пользователей или товаров:
– Тривиальные рекомендации;
– Подход на основе сходств пользователей (user-based collaborative filtering);
– Подход на основе сходств товаров (item-based collaborative filtering).

1.3.5 Тривиальные рекомендации

Пусть пользователю u_o понравился товар i_0. Найдем множество пользователей, которым также понравился этот товар:

$$U(i_0) = \{u \in U | \exists r_{ui_0}\}$$

Далее среди товаров, купленных пользователями из множества $U(i_0)$ найдем те, которые наиболее похожи на исходный товар:

$$I(i_0) = \{i \in I | sim(i_0, i) > \alpha \quad \exists r(u, i), u \in U(i_0)\}$$

Пользователю рекомендуется k товаров из множества $I(i_0)$ с максимальным значением $sim(i_0, i)$.

У такого подхода есть ряд проблем. Так, например, все рекомендации будут тривиальные, то есть в основном будут рекомендоваться популярные товары. Также не учитываются интересы конкретного пользователя u_0. Однако, всегда можно будет порекомендовать конкретный объект нетипичным пользователям, несмотря на то, что это будет популярный товар.

1.3.6 Подход на основе сходств пользователей. User-based collaborative filtering

Рассмотрим второй подход. Классическая реализация алгоритма основана на принципе k ближайших соседей. Для каждого пользователя ищется k наиболее похожих на него (в терминах предпочтений) и дополняется информацией о пользователе известными данными по его соседям. Допустим, необходимо сделать рекомендации для фиксированного пользователя u_0. Найдем множество $U(u_0)$ пользователей, похожих на данного:

$U(u_0) = \{u \in U | sim(u_0, u) > \alpha\}$, где α заданный порог схожести.

После этого для каждого товара вычислим, как часто пользователи из множества $U(u_0)$ покупали его:

$p_i = \frac{|\{u \in U(u_0) | \exists r_u i\}|}{|U(u_0)|}$

Пользователю рекомендуется k товаров с максимальными значениями p_i. Таким образом, будет вероятность рекомендовать не только самые популярные товары, но и те, что популярны среди данного типа пользователей. Также в таком подходе учитываются интересы пользователя u_0.

Проблема данного подхода заключается в том, что он позволяет строить рекомендации только в том случае, если для данного пользователя существуют похожие на него. Если же пользователь новый или нетипичный, то подобрать что-либо не получится. У такой реализации алгоритма существует недостаток – он плохо применим на практике из-за квадратичной сложности. Действительно, при реализации метода ближайшего соседа, требуется расчет всех попарных расстояний между пользователями (их может быть миллионы).

1.3.7 Подход на основе сходств товаров. Item-based collaborative filtering

Последний подход основан на сходстве товаров. Подход Item-based является альтернативой классическому подходу User-based, описанному выше, и почти полностью его повторяет, за исключением одного момента — применяется он к транспонированной матрице предпочтений, то есть ищет близкие товары, а не пользователей. Для него определяется множество товаров, похожих на те, которые интересовали пользователя u_0:

$I(u_0) = \{i \in I | \exists r_{u_0 i_0}, sim(i_0, i) > \alpha\}$, где α заданный порог схожести

Далее для каждого товара из найденного множества вычисляется его сходство с товарами из множества пользователя:

$p_i = max i_0 : \exists r_{u_0 i_0}, sim(i_0, i)$

Пользователю рекомендуется k товаров с наибольшими значениями p_i. В таком подходе решается проблема нетипичного пользователя, так как необязательно иметь пользователей со схожими интересами, и подход позволяет найти товары, похожие на интересные ему. Однако, существуют и проблемы: есть вероятность, что вместо действительно интересных товаров будут рекомендоваться популярные.

При использовании item-based подхода рекомендации имеют свойство быть консервативными. Действительно, разброс рекомендаций получается меньше и, следовательно, меньше вероятность показать нестандартные товары. В item-based подходе сложность вычислений снижается, так как необходимо хранить матрицу

схожести товаров, а в большинстве случаев, товаров меньше, чем пользователей и соответсвенно размер матрицы будет меньше. Оценка близости товаров более точная, чем оценка близости клиентов. Этот факт следует из того, что пользователей обычно в разы больше, чем товаров и, следовательно, стандартная ошибка при расчете корреляции товаров там существенно меньше. У алгоритма существенно больше данных, чтобы сделать вывод. Предпочтения пользователя могут меняться со временем, но описание товаров гораздо более устойчивы. В user-based варианте матрица сильно разрежена, то есть преобладают нулевые элементы. Из-за этого список товаров, которые в итоге можно порекомендовать, получается очень небольшим.

Другое потенциально возможное улучшение — при вычислении item similarity взвешивать пользователей. Например, с увеличением оценок, сделанными пользователем, будет увеличиваться его вес при сравнении двух товаров.

1.3.8 Контентные модели. Content-based recommendations

В коллаборативной фильтрации учитывается информация о вкусах пользователей и об их сходствах, но при этом никак не используется свойства самих пользователей и товаров. При этом может быть полезным находить товары, похожие описанием на те, которыми пользователи интересовались. В рамках этого подхода описание товара (content) сопоставляется с интересами пользователя, полученными из его ранее известных оценок. Чем больше товар этим интересам соответствует, тем выше оценивается потенциальная заинтересованность пользователя. Очевидное требование здесь — у всех товаров в каталоге должно быть описание. Особенно видна польза такого подхода в рекомендательных системах, где пользователям предлагают видео, музыку или статьи. Скорее всего пользователю, любящему электронную музыку, захочется послушать не только то, что нравится другим электронщикам, но и то, что по звучанию похоже на его любимых исполнителей.

При таком подходе все товары описываются векторами, называемыми эмбедингами (embeddings). Затем измеряется сходство между вектором нового товара и векторами товаров из истории пользователя: можно вычислять как минимальное, так и среднее расстояние до векторов из истории. Однако, не все элементы необходимо включать в модель: например, союзные слова, не несут никакой смысловой нагрузки и могут быть удалены. Поэтому при определении числа совпадающих элементов в двух векторах все измерения нужно предварительно взвешивать по их значимости. Данная задача решается преобразованием TF-IDF, которое ставит больший вес более редким интересам. Совпадение таких интересов имеет большее значение при определении близости двух векторов, чем совпадение популярных.

Очевидно, существует множество методов учета данных о пользователях и товарах, но никогда нельзя предсказать заранее, какой из них подойдет в данной

задаче. При формировании вектора представления товара, вместо единичных слов можно использовать n-граммы (пары слов, тройки и т.д.). Такое дополнение сделает модель более точной, однако, потребуется больше данных и времени для обучения. Описания товара от можно взвешивать по-разному. Например, давать больший вес активным пользователям, у которых много оценок. Аналогично такой подход можно применить и к товару. Чем больше средний рейтинг, тем больше его вес.

1.4 Проблематика алгоритмов рекомендательных систем

Все перечисленные выше подходы имеют такие проблемы, как:

— Не хватает теоретического обоснования;

— Необходимо хранить всю матрицу рейтингов R;

— Проблема холодного старта.

Существует множество способов измерить схожесть пользователей и товаров. Например, корреляция Спирмана, косинусное расстояние. Также существует большое множество гибридных методов, учитывающих преимущества подходов. Но без теоретического обоснования подходов сложно точно определить, какой подход где лучше использовать. Обычно, ответ на этот вопрос можно узнать только на практике.

Проблема холодного старта заключается в том, что накоплено недостаточное количество данных для корректной работы рекомендательной системы. Например, непонятно, что рекомендовать новоприбывшему пользователю или кому нужно рекомендовать новый или редко покупаемый товар. От этой проблемы страдают большинство подходов в рекомендательной системе, так что эта проблема остается до сих пор актуальной. Проблема холодного старта также актуальна и для неперсонализированных рекомендаций. Общий подход здесь — заменять то, что в данный момент не может быть посчитано, различными эвристиками (например, заменять средним рейтингом или не использовать товар, пока не соберутся данные).

Единственная действительно разрешимая проблема — необходимость хранить всю матрицу рейтингов R. Существуют подходы, помогающие уменьшить размерность матрицы.

Помимо проблем у рассмотренных подходов есть и преимущества:

— Легко понять;

— Легко реализовать.

Благодаря этим сильным плюсам, зачастую memory-based подходы используют в качестве baseline.

1.5 Функционал ошибки

Как говорилось ранее, еще один важный пункт, который следует установить в любой задаче машинного обучения, это функционал ошибки. Существует довольно большое множество метрик качества рекомендательных систем. Различают онлайн-метрики и оффлайн-метрики.

Онлайн-метрика — метрика, ключевая с точки зрения бизнеса. Например, это может быть среднее время, проведенное пользователем на сайте, или средний чек пользователя. Затем выбирают оффлайн-метрику или линейную комбинацию оффлайн-метрик, которая лучше всего коррелирует с выбранной бизнес метрикой.

Под онлайн-метрикой понимается показатель, который можно измерить только при запуске рекомендательной системы на реальных пользователя, а под оффлайн-метрикой — показатель, который можно измерить для модели на исторических данных. Так же иногда пытаются найти промежуточную метрику, которая коррелирует с основной, но при этом быстро реагирует на изменения в работе рекомендательной системы.

1.5.1 Качество предсказания событий

Рассмотрим несколько оффлайн-метрик. Поскольку модель обучается для предсказывания r_{ui}, логично оценивать качество решения именно этой задачи.

Если модель предсказывает рейтинг, время нахождения на странице или любую другую вещественную величину, то качество рекомендаций может быть измерено с помощью MSE, RMSE, MAE или другие регрессивных метрик.

Если модель предсказывает вероятность клика на статью, покупки товара, лайка или наступления любого другого события, то качество можно измерить с помощью метрик качества классификации: доля правильных ответов, точность, полнота, F-мера, AUC-ROC, log-loss.

Если вспомнить, что мы показываем пользователю только k товаров, получивших самые высокие предсказания модели, то можно понять, что интересует качество только для этих товаров. Если через $R_u(k)$ обозначить предсказание для пользователя и k товаров, а через L_u - товары, для которых действительно произошло интересующее нас событие, то можно ввести следующие метрики:

Наличие верной рекомендации:

$$hitrate@k = [R_u(k) \cap L_u \neq 0]$$

Точность:

$$precision@k = \frac{|R_u(k) \cap L_u|}{|R_u(k)|}$$

Полнота:

$$recall@k = \frac{|R_u(k) \cap L_u|}{|L_u|}$$

1.5.2 Покрытие пользователей

Сложно говорить о том, насколько хорошо работает рекомендательная система, основываясь только на качестве предсказаний. Возможно, пользователь купил бы товары, предсказанные моделью, и без её помощи. Поскольку нельзя узнать, повлияло ли предсказание модели на поведение пользователя, имеет смысл анализировать и другие метрики качества, которые могут косвенно говорить о пользе предсказаний модели.

Важно следить за долей пользователей, для которых рекомендательная система не может ничего порекомендовать. Такие ситуации могут происходить, например, из-за отсутствия определенных признаков в модели или из-за низкой уверенности модели. В таком случае необходимо отслеживать проблемы с покрытием в модели рекомендаций.

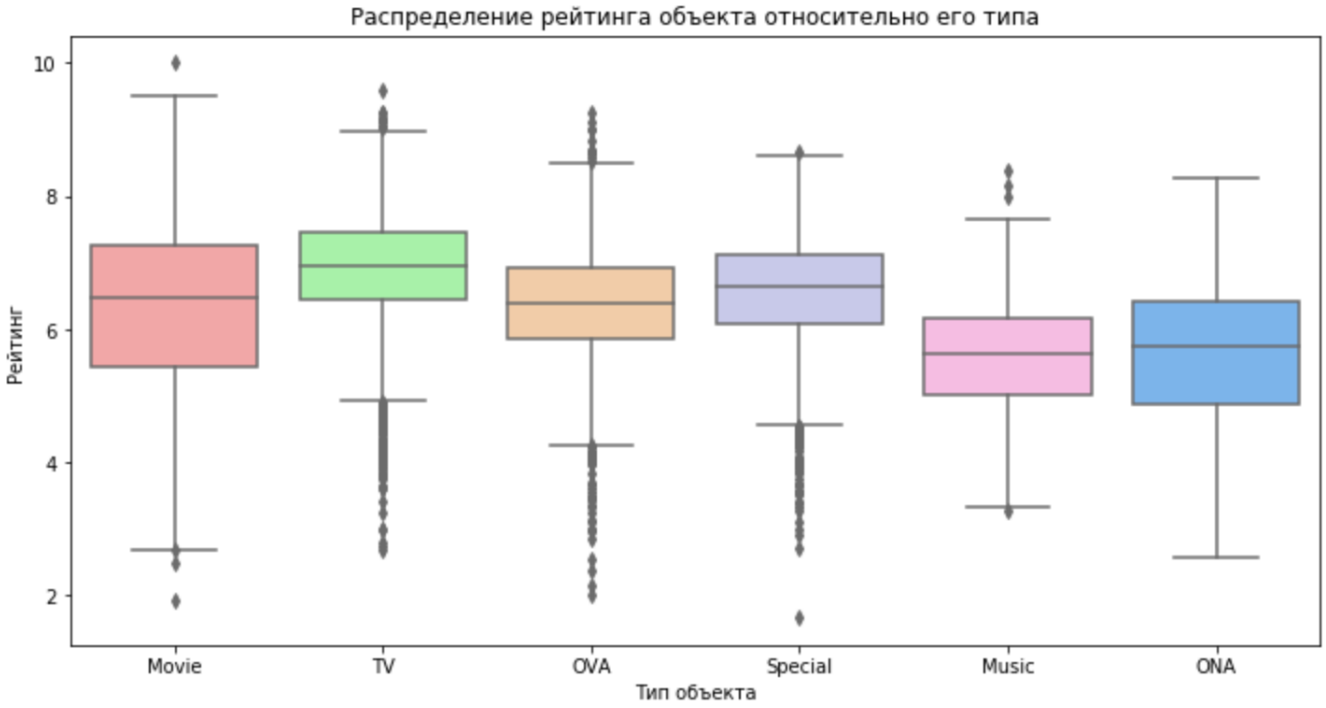
2 Практическая часть

2.1 Работа с данными

Для построения алгоритмов рекомендательной системы был выбран датасет с информацией о предпочтениях ≈ 74 тысяч пользователей в просмотре аниме-сериалов. Каждый пользователь имеет возможность добавить сериал в список просмотренного и дать ему оценку от 1 до 10, где 10 значит, что пользователю понравился сериал. В данных содержатся рейтинги каждого пользователя для всех просмотренных им объектов в виде: user id — уникальный идентификационный ключ пользователя; anime id — уникальный идентификационный ключ просмотренного пользователем сериала; rating — рейтинг пользователя. Также в данных представлена информация о самих сериалах в виде: anime id; name — название сериала; genre — список жанров, к которому можно отнести объект; type — фильм, сериал, дополнительные материалы (в работе будет рассматриваться только тип сериал, так как таких данных большинство); episodes — количество эпизодов; rating — средний рейтинг пользователей; members — количество пользователей, посмотревших сериал.

Был проведен первичный анализ данных средствами Python3 и его библиотек: matplotlib, seaborn, pandas.

Из круговой диаграммы видно, что, в процентном соотношении, в данных содержится больше информации об объектов с типом "TV" или, проще говоря, о сериалах.



Используя диаграмму размаха можно заметить, что медиана рейтинга для объектов с типом "TV" составляет ориентировочно 7 из 10 и является максимальной среди всех типов. Также можно заметить, что разброс первого и третьего квартиля, длина усов почти не отличаются от объекта типа "OVA", только находятся выше, что может говорить о более высоких оценках.

Исходя из анализа графиков имеет смысл рассматривать только объекты типа "TV", то есть сериалы, так как данных о них больше и средний рейтинг выше, чем у остальных мультипликаций.

	episodes	rating	members
count	3570.000000	3671.000000	3.671000e+03
mean	35.969468	6.902299	4.377497e+04
std	80.722257	0.863526	9.023887e+04
min	2.000000	2.670000	1.200000e+01
25%	12.000000	6.440000	4.740000e+02
50%	24.000000	6.940000	6.227000e+03
75%	39.000000	7.460000	4.597250e+04
max	1818.000000	9.600000	1.013917e+06

Всего в датасете 3671 уникальных аниме-сериалов. Из таблицы видно, что у некоторых отсутствует информация о количестве эпизодов, таких меньше 100, поэтому на дальнейшие расчеты это влиять не будет. Средний рейтинг сериала действительно оказался близким к 7, он составляет 6.9. Маленькое значение среднеквадратического отклонения 0.864 означает, что значения рейтинга

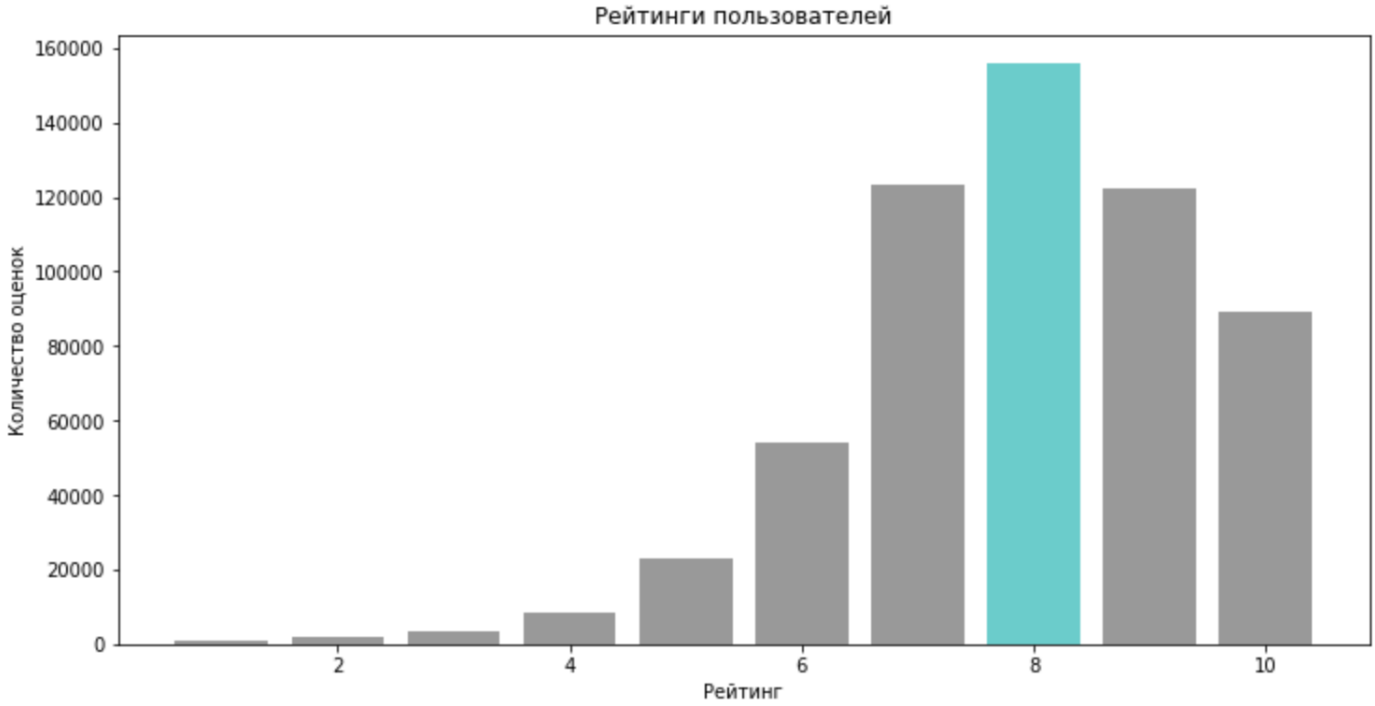
сгруппированы вокруг среднего.

Из графика зависимости среднего рейтинга от количества пользователей, посмотревших сериал, можно сделать вывод, что чем больше просмотров, тем больше положительных оценок и, в итоге, рейтинг сериала оказывается выше среднего. Следовательно, при построении рекомендательной системы не стоит опираться на рейтинг сериала, так как предложения алгоритма пользователю будут состоять из популярных мультипликаций и не будут основаны на его предпочтениях. Предпочтения пользователя определяются его собственными оценками.

Корреляционная матрица подтверждает зависимость между рейтингом и количеством просмотров.

Итак, исходя из проделанного анализа данных, было принято решение

использовать предыдущие оценки пользователя для рекомендаций.



Чаще всего пользователи ставят оценку 8.

2.2 Разбиение выборки на обучающую и тестовую

В любой задаче машинного обучения набор данных разбивается на две выборки: обучающую (train) и тестовую (test). Модель обучают на train выборке, а на test выборке модель проверяют, то есть получают некоторое предсказание. Это необходимо для того, чтобы оценить качество работы алгоритма на уже имеющихся данных и избежать переобучения/недообучения модели.

Существуют несколько способов разбиение данных на выборки. В работе используется разбиение функцией train_test_split из библиотеки Sklearn. Функция разбивает данные рандомным способом, что не всегда бывает корректным методом разбиения, но на данном этапе разработки рекомендательной системы этот метод подходит.

	user_id	name	user_rating
1	3	Naruto	8
2	5	Naruto	6
5	21	Naruto	8
6	28	Naruto	9
7	34	Naruto	9

Так выглядит часть train датасета. Заметим, что название объектов имеют тип строка. Такой тип данных не подходит для алгоритма, поэтому каждому уникальному названию необходимо сопоставить уникальный номер.

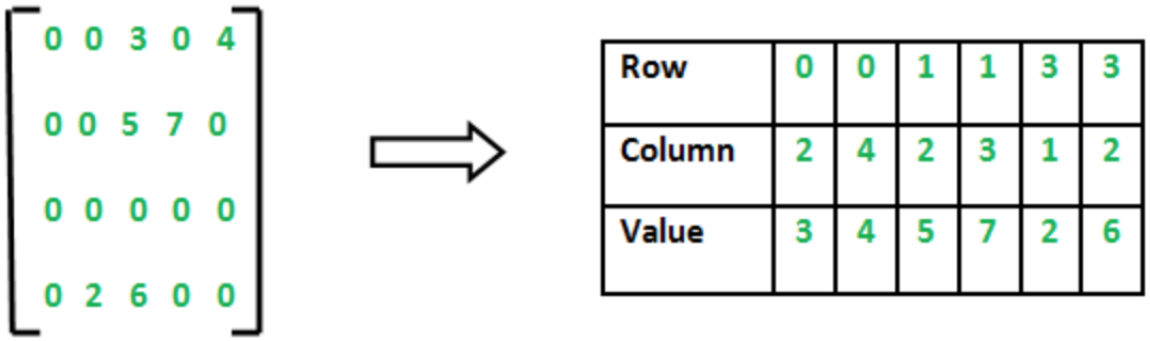
	user_id	name	user_rating
1	0	0	8
2	1	0	6
5	2	0	8
6	3	0	9
7	4	0	9

Итоговый вид выборки.

2.3 Создание матрицы рейтингов

Составим матрицу рейтингов R размерности $N \times M$, где N — количество пользователей, а M — количество сериалов. В матрице R, по одной из осей отложены клиенты сервиса (Users), а по смежной – сериалы (Items). На пересечении некоторых пар (user, item) эта матрица заполнена оценками пользователя сериалу (Ratings). Если в такой матрице хранить все значения, то есть не только сами оценки пользователей, но так же и нули, говорящие о том, что оценки не было, то в итоге такая матрица займет слишком много места, что увеличит время работы алгоритма и затраты памяти компьютера. Матрица R будет считаться разреженной матрицей, то есть матрицей с преимущественно нулевыми элементами.

Для того, чтобы уменьшить матрицу рейтингов создается матрица без нулевых значений, но с сохранением первоначальных координат ячеек. При создании такой матрицы, достаточно знать координаты каждого ненулевого элемента. Координатами являются id пользователей и id аниме.



Пример преобразования матрицы R.

2.4 Baseline prediction

Для начала попробуем предсказывать рейтинги пользователей простым способом. Найдем средний рейтинг для всех пользователей и всех сериалов. Для этого просуммируем все проставленные рейтинги и поделим на их количество:

$\mu = \frac{1}{n} \sum_{u,i} r_{ui}$

Уже можно было бы сказать, что неизвестная нам оценка, которую поставит некий пользователь аниме-сериалу, равна среднему. Но это будет некорректное

предположение, так как при этом не учитываются данные о пользователе. Будем учитывать его предвзятость и переоценку/недооценку аниме-сериалу. Например, некоторые пользователи ставят в основном 10, а 9 для них - ужасный сериал; для некоторых пользователей наблюдается обратная ситуация: даже 5 для них — это высокая оценка.

Для того, чтобы вычислить "сдвиг" в оценках пользователя u относительно средней оценки μ, достаточно вычесть из всех его оценок r_{ui} среднюю оценку μ, просуммировать результат и поделить на количество оценок $|I_u|$, который этот пользователь поставил. Так как не все пользователи ставили оценки, то был введен параметр α, чтобы избежать деления на 0. Параметр α можно взять равным 1, но так как α является также и коэффициентом сглаживания, то было взято значение, равное 25.

$b_u = \frac{1}{|I_u|+\alpha} \sum_{i \in I_u} (r_{ui} - \mu), \qquad \alpha = 25$

Аналогично вычислим "сдвиг" в оценивании аниме i, но при вычислении учтем еще и найденные ранее сдвиги в оценках пользователей. Также введем коэффициент сглаживания β и примем его равным коэффициенту α.

$b_i = \frac{1}{|U_i|+\beta} \sum_{u' \in U_i} (r_{u'i} - b_{u'} - \mu), \qquad \beta = 25$

Итоговое предсказание оценки r_{ui} пользователя u объекту i:

$\hat{r}_{ui} = \mu + b_u + b_i$

Средняя оценка пользователей равна 7.882. С учетом средней оценки вычислим "сдвиг" в оценке для каждого пользователя по формуле, описанной выше.

```
[-0.13464002632911784, -3.2868801654863478,
 -0.3677503990118461, ..., 0.042981586668341655,
 0.11981638913902837, 0.08144312512988011],
```

Из матрицы сдвигов оценок видно, что, например, второй пользователь в среднем ставит оценку на 3 баллы меньше, чем средняя оценка всех пользователей. Это значит, что сериалу, которому обычный пользователь поставил бы оценку 8, наш поставит всего лишь 5. Именно из-за возникновения таких ситуаций необходимо делать поправку на предвзятость.

Средняя оценка сериала равна 6.9. Также вычислим "сдвиг" для объектов рекомендации.

```
[-0.09248308250339068, 0.3791027226665885,
 -0.27043095731978617, ..., -0.00019449975560034908,
 0.03826703870593811, -0.03865603821713881],
```

В итоге объединим "сдвиги" пользователей и "сдвиги" объектов в единую матрицу. Тогда значение b_{ui}, находящееся на пересечении строки u и столбца i, будет равно "сдвигу" оценивания аниме i пользователем u относительно средней оценки μ.

Сложим среднюю оценку μ и все известные нам сдвиги B в соответствии с формулой, описанной выше, и получим простую рекомендательную систему. Значение r_{ui}, находящееся на пересечении строки u и столбца i матрицы

получившейся матрицы — оценка, которую поставит пользователь u аниме-сериалу i.

```
[7.655355637790608, 8.126941442960588, 7.477407762974213, ...,
 7.747644220538398, 7.786105758999938, 7.70918268207686],
[4.503115498633378, 4.974701303803357, 4.3251676238169825, ...,
 4.595404081381169, 4.633865619842707, 4.55694254291963],
[7.42224526510788, 7.89383107027786, 7.244297390291485, ...,
 7.51453384785567, 7.552995386317209, 7.476072309394132],
...,
[7.832977250788068, 8.304563055958047, 7.655029375971672, ...,
 7.925265833535859, 7.963727371997397, 7.88680429507432],
[7.909812053258754, 8.381397858428734, 7.731864178442359, ...,
 8.002100636006546, 8.040562174468084, 7.963639097545006],
[7.8714387892496065, 8.343024594419585, 7.693490914433211, ...,
 7.963727371997397, 8.002188910458935, 7.925265833535859]],
```

Посмотрим работу алгоритма на примере пользователя с $id = 9910$.

```
[('Clannad: After Story', 9.115976535983961),
 ('Mushishi Zoku Shou 2nd Season', 9.029134111950487),
 ('Yojouhan Shinwa Taikei', 9.001924837850023),
 ('Haikyuu!!: Karasuno Koukou VS Shiratorizawa Gakuen Koukou',
  8.970470912062122),
 ('Shigatsu wa Kimi no Uso', 8.967962954893263),
 ('Monster', 8.960735472341119),
 ('Haikyuu!! Second Season', 8.948307261905825),
 ('Mushishi', 8.941224547248364),
 ('Monogatari Series: Second Season', 8.907109629348938),
 ('Mushishi Zoku Shou', 8.90328163554299)]
```

Функция выдает 10 рекомендаций в виде: название сериала, прогнозируемая оценка пользователя. В тестовой выборке, которая не учитывалась при построении прогноза, есть сериал "Shigatsu wa Kimi no Uso", который пользователь оценил в 8 баллов. Предсказание алгоритма было 8.97. Порядок рейтинга предсказан довольно точно.

2.5 Использование схожести пользователей

В предыдущем алгоритме предсказывали рейтинг пользователей, отталкиваясь от среднего рейтинга всех пользователей. Поэтому, даже если пользователю u нравится только жанр ужасы, то при вычислении его оценок также учитывались оценки тех пользователей, кто больше любит жанр комедия, что неправильно. Необходимо модифицировать алгоритм Baseline prediction: во-первых, нужно учитывать только N похожих пользователей (множество U_{sim}); во-вторых, каждый рейтинг будет учитывать с весом, равным схожести пользователей.

$$\hat{r}_{ui} = \frac{\sum_{u' \in U_{sim}} sim(u,u') r_{u'i}}{\sum_{u' \in U_{sim}} |sim(u,u')|}$$

Немного улучшим алгоритм: усредним не оценки похожих пользователей, а их "сдвиги" в оценках, и добавим это к средней оценке пользователя u.

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{u'} sim(u,u')(r_{u'i} - \bar{r_{u'}})}{\sum_{u'} |sim(u,u')|}$$

Здесь $\bar{r}_u$ - средний рейтинг пользователя u.

Используем косинусную близость:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Для облегчения задачи нормализуем вектора рейтингов для каждого пользователя.

$R' = r_{ui} - \mu - b_u - b_i$.

Реализована функция, которая возвращает топ 10 аниме-сериалов, похожих по аудитории. Стоит отметить, что схожесть вычисляется не по жанру, а по группам пользователей.

```
Top 10 to: Naruto
#1: Bleach
#2: Dragon Ball GT
#3: Dragon Ball Z
#4: Yu☆Gi☆Oh! Duel Monsters
#5: Death Note
#6: Sword Art Online
#7: Hello! Lady Lynn
#8: Fairy Tail
#9: Pokemon Advanced Generation
#10: Fullmetal Alchemist
```

Алгоритм вернул топ 10 сериалов, похожих на "Naruto". Замечу, что рекомендованные мультипликации, как и первоначальный сериал, ориентированы на зрителей 12+, поэтому аудитория у них, действительно, схожа.

Использую матрицу близости пользователей, найдем самых близких N к пользователю с $id = 9910$ и по ним определим рейтинг.

```
[('Baccano!', 0.03395301276493162),
 ('Fullmetal Alchemist', -0.03378985656271696),
 ('Clannad: After Story', -0.07279388363607137),
 ('Dragon Ball Z', -0.12646549150424596),
 ('Ghost in the Shell: Stand Alone Complex', -0.13791327036770473),
 ('Log Horizon', -0.14305799315168718),
 ('So Ra No Wo To', -0.1457548940346726),
 ('Dragon Ball', -0.14812219300326257),
 ('Gosick', -0.15099667787443327),
 ('Princess Tutu', -0.15178070277973654)]
```

Как и в первом случае, сравним полученные рекомендации с test частью выборки. Сериалу "Baccano!", "Clannad: After Story" пользователь поставил 9/10. Следовательно, рекомендации действительно обоснованы.

2.6 Латентные признаки: SVD

Есть возможность использовать для предсказания знания о пользователе, которые он сам сообщил. Это может быть пол, возраст, любимые фильмы. Но что делать, если пользователь ничего не указал? В этом случае помогает использование латентных (скрытых) признаков. Например, на основе анализа оценок пользователей можно заметить, что User 1 предпочитает больше мелодрамы, тогда как User 2 любит больше экшн фильмы, хотя ни тот, ни другой нигде это не указали. Это и есть латентные признаки.

Один из способов "достать" такие признаки - разложение (факторизация) матрицы

$R = UI^T$,

где $R \in \mathbb{R}^{u \times i}$, $U \in \mathbb{R}^{u \times k}$ и $I \in \mathbb{R}^{i \times k}$. Такого разложения можно добиться, используя SVD разложение.

Пример работы алгоритма SVD:

```
[('Cowboy Bebop', 9.29284029905841),
 ('Gintama°', 9.222820891226876),
 ('Tengen Toppa Gurren Lagann', 9.193915135838783),
 ('Fullmetal Alchemist: Brotherhood', 9.173117975607369),
 ('Haikyuu!!: Karasuno Koukou VS Shiratorizawa Gakuen Koukou',
  9.153435902838636),
 ('Clannad: After Story', 9.13874961294318),
 ('Mahou Shoujo Madoka★Magica', 9.11096707768818),
 ('Neon Genesis Evangelion', 9.077689082545925),
 ('Steins;Gate', 9.04447477323739),
 ('Gintama&#039;', 9.011997178619948)]
```

Заключение

В работе были проанализированы первичные данные и на основе анализа были сделаны корректировки и выводы о том, как эффективнее использовать данную информацию при написания алгоритмов рекомендательной системы.

Были реализованы три метода предсказания товара, который понравится пользователю. Baseline prediction был самым простым и понятным в реализации. С его помощью некоторому определенному пользователю был порекомендован сериал, который пользователь оценил в 8 баллов из 10 при предсказанном рейтинге 8.97.

Используя схожесть пользователя для того же клиента онлайн сайта, было предсказано два сериала, оценка которых в конечном итоге составила 9/10.

Проблематика полной оценки работы алгоритмов заключается в нехватке данных о пользователе: нельзя сказать, какой рейтинг поставил клиент получившимся рекомендациям. Однако, факт того, что в некоторых случаях предсказанная и реальная оценка совпадали, дает право сделать вывод о том, что построенная рекомендательная система работает.

Работу алгоритмов можно улучшить, обработав больший массив данных и посмотрев метрики для рекомендации сериалов в онлайн режиме.

Список использованных источников

[1] Yehuda Koren, Robert Bell and Chris Volinsky. MATRIX FACTORIZATION TECHNIQUES FOR RECOMMENDER SYSTEMS - 2009.

[2] Michael D. Ekstrand, John T. Riedl and Joseph A. Konstan. Collaborative Filtering Recommender Systems - Foundations and Trends in Human-Computer Interactions Vol. 4, No. 2 (2010) p. 81–173.

[3] Фальк Ким. Рекомендательные системы на практике - ДМК-Пресс, 2020 г.

[4] Маккини Уэс. Python и анализ данных - ДМК-Пресс, 2015 г.

Приложение

Исходный код можно найти по ссылке.