Изображение выглядит как эмблема, символ, герб, нашивка

Автоматически созданное описание

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА №1**

**по дисциплине**

**«Разработка обеспечивающих подсистем систем поддержки принятия решений»**

Студент группы:ИКБО-04-22 \_\_Кликушин В.И.\_ *(Ф. И.О. студента)*

Преподаватель \_\_ Гуличева А.А.\_\_

*(Ф.И.О. преподавателя)*

|  |  |  |
| --- | --- | --- |
|  |  |  |

Москва 2025

# СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc211426541)

[1 ПОСТАНОВКА ЗАДАЧИ 5](#_Toc211426542)

[2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 6](#_Toc211426543)

[2.1 Постановка задачи 7](#_Toc211426544)

[2.2 Анамнестические методы 7](#_Toc211426545)

[2.3 Математические меры сходства 8](#_Toc211426546)

[3 ДОКУМЕНТАЦИЯ К ДАННЫМ 11](#_Toc211426547)

[3.1 Описание предметной области 11](#_Toc211426548)

[3.2 Анализ данных 11](#_Toc211426549)

[3.3 Предобработка данных 15](#_Toc211426550)

[4 ПРАКТИЧЕСКАЯ ЧАСТЬ 16](#_Toc211426551)

[4.1 Функциональные возможности 16](#_Toc211426552)

[4.2 Рекомендательная система 20](#_Toc211426553)

[ЗАКЛЮЧЕНИЕ 23](#_Toc211426554)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 24](#_Toc211426555)

[ПРИЛОЖЕНИЯ 25](#_Toc211426556)

# ВВЕДЕНИЕ

Современные информационные системы ежедневно обрабатывают огромное количество данных о пользователях и их взаимодействиях с цифровым контентом. В условиях растущего объёма данных проблема предоставления релевантной информации отдельному пользователю становится одной из ключевых. Пользователи интернет-магазинов, медиасервисов или онлайн-платформ регулярно сталкиваются с ситуацией информационной перегрузки: выбор нужного товара или контента вручную требует значительных временных и когнитивных ресурсов. Рекомендательные системы решают эту задачу, автоматически предлагая пользователям контент, который с наибольшей вероятностью будет им интересен, на основе анализа их собственных предпочтений и поведения других пользователей.

Актуальность данной темы обусловлена широким практическим применением рекомендательных систем в различных областях: от электронной коммерции и стриминговых сервисов до социальных сетей и образовательных платформ. Согласно исследованиям, внедрение эффективных рекомендательных алгоритмов позволяет существенно повысить вовлечённость пользователей и конверсию — до 35% от общего объёма продаж на платформах, таких как Amazon и Netflix, формируется за счёт рекомендаций. Помимо практической значимости, рекомендательные алгоритмы представляют собой интересный объект с точки зрения анализа данных, поскольку предполагают работу с разреженными матрицами, выбор мер сходства и реализацию различных стратегий прогнозирования предпочтений.

В данной работе основное внимание уделяется анамнестическим методам коллаборативной фильтрации, которые предполагают поиск соседей — схожих пользователей (user-based) или схожих объектов (item-based) — и использование их предпочтений для прогнозирования неизвестных оценок.

Для вычисления степени сходства между пользователями и объектами в литературе предлагаются различные меры: коэффициент корреляции Пирсона, косинусное сходство (Отиаи), коэффициент Жаккара и Lp-нормы. В работе реализованы и сравниваются несколько таких мер, что позволяет эмпирически оценить их влияние на точность рекомендаций. Для оценки качества используются стандартные метрики ошибок предсказания — RMSE и MAE.

Таким образом, целью данной работы является реализация и сравнительный анализ анамнестических методов рекомендательных систем на основе пользовательских и предметных соседств, с использованием нескольких математических мер сходства. Для достижения этой цели в работе выполнены следующие этапы: анализ предметной области и структуры данных, реализация алгоритмов user-based и item-based коллаборативной фильтрации с различными метриками, а также оценка и визуализация результатов с помощью статистических показателей точности.

# 1 ПОСТАНОВКА ЗАДАЧИ

Цель работы: приобрести навыки реализации анамнестических коллаборативных методов рекомендательных систем, основанных на соседстве пользователей и объектов.

Задачи: создать программную реализацию анамнестического метода рекомендательной системы (РС), основанной либо на соседстве пользователей, либо на соседстве элементов (предметов), включающую актуальную предметную область для применения РС (вроде маркетплейса, медиа ресурсов, соц. сетей, экономической сферы и т. д.), математические меры сходств: расстояние Жаккара, норму лебегова пространства (Lp-норму), коэффициент Отиаи, коэффициент корреляции Пирсона и т.п., необходимо реализовать не менее 3-ёх любых методов (пояснить, почему были выбраны именно эти методы), сравнение вышеописанных методов и, исходя из результатов, выбор наиболее подходящего(-их) метода(-ов) для решения задачи.

# 2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Рекомендательная система подбирает и предлагает пользователю, релевантный контент, основываясь на своих знаниях о пользователе, контенте и взаимодействии пользователя и контента. Рекомендательная система стремится подобрать и предложить контент, который пользователь ещё не видел и который наиболее вероятно будет ему интересен, то есть это система прогнозирования предпочтений. Рекомендательные системы представляют собой программный комплекс, который определяет и распознает интересы и предпочтения пользователей, формируя рекомендации в соответствии с ними.

В контексте рекомендательных систем наиболее значимы следующие определения:

1. Прогноз – это предположение, насколько пользователю понравится контент.
2. Релевантность – расположение контента в соответствии с тем, что всего подходит пользователю в данный момент. Релевантность сочетает в себе контекст, демографические данные и (ожидаемые) оценки.
3. Рекомендации – лидеры по релевантности.
4. Персонализация - сочетание релевантности и наглядности.

Рекомендаторы используют данные о контенте, о пользователях и о взаимодействиях пользователей и контента. Такие рекомендации являются персональными, так как основаны на персональных предпочтениях и формируются специально для данного пользователя.

В базовых подходах для рекомендательных систем могут использоваться два вида данных: информация о взаимодействии пользователей с объектами интереса и информация, предоставленная самими пользователями, например, атрибуты, указанные в профиле или релевантные ключевые слова.

Первую группа методов чаще всего называют методами коллаборативной фильтрации, для методов второй группы обычно используется название рекомендаций на основе контента.

## 2.1 Постановка задачи

Имеются следующие данные:

* множество пользователей (*users*, *u* ∈*U*);
* множество объектов (*items*, *i* ∈*I*);
* множество событий (действия, которые пользователи совершают с объектами) (*events*, (𝑢, 𝑖, ) ∈ *D*);

Событие описывается так: пользователь *u* поставил оценку объекту *i*.

Требуется:

* предсказать оценку объекту, которого пользователь ещё не видел (Формула 2.1).

(2.1)

* вычислить персональные рекомендации для пользователя *u* (Формула 2.2).

(2.2)

## 2.2 Анамнестические методы

Фильтрация в окрестности может быть реализована двумя методами: user-based (юзер-бейсд – сходства пользователей) и item-based (айтем-бейсд – сходства элементов), они основаны на построении матриц схожести.

В общем задача нахождения схожести может быть определена следующим образом: имеется два элемента и ; сходство между ними определяются функцией . Возвращаемое этой функцией значение пропорционально степени сходства между элементами. Тогда для идентичных элементов 𝑠, а для элементов, не имеющих ничего общего . Изменение сходства тесно связано с расчетом различия между элементами. Математически это можно выразить так: Сходство = 1 – Различие.

Целью обоих направлений user-based и item-based является выделение схожих объектов в группы на основе матрицы оценок. В первом случае определяется сходство пользователей: найти других пользователей, чьи прошлые оценки поведения похожи на те, что и у текущего пользователя, и использовать их оценки других элементов для прогнозирования предпочтения текущего пользователя. Второй подход, на основе сходства элементов, в этом случае вместо того, чтобы использовать подобие между поведением пользовательских оценок для прогнозирования предпочтения, используется сходство между оценками моделей элементов. Если два элемента, как правило, имеют одинаковые оценки пользователей, то они похожи, и пользователи должны иметь аналогичные предпочтения для подобных элементов.

Для определения сходства между пользователями или элементами используют различные подходы.

## 2.3 Математические меры сходства

1. Расстояние Жаккара.

Этот параметр называется коэффициентом сходства Жаккара, который показывает на сколько похожи два набора данных. Коэффициент Жаккара измеряет подобие между конечными множествами выборок, и определяется как размер пересечения, деленного на размере объединения множеств выборок (Формула 2.3).

(2.3)

1. -норма.

Расстояние также известно, как расстояние городских кварталов, манхэттенское расстояние, расстояние такси, метрика прямоугольного города — оно измеряет дистанцию не по кратчайшей прямой, а по блокам. Название «манхэттенское расстояние» связано с уличной планировкой Манхэттена. Манхэттенское расстояние вычисляется по Формуле 2.4.

(2.4)

1. -норма.

-норму иначе называется евклидовым расстоянием. Евклидова метрика (евклидово расстояние) – метрика в евклидовом пространстве – расстояние между двумя точками евклидова пространства, вычисляемое по теореме Пифагора. Для векторов и евклидово расстояние определяется по Формуле 2.5.

(2.5)

Евклидова метрика — наиболее естественная функция расстояния, возникающая в геометрии, отражающая интуитивные свойства расстояния между точками.

1. Коэффициент Отиаи.

Коэффициент Отиаи (косинусный коэффициент, косинусное подобие) – бинарная мера сходства, предложенная японским биологом Акирой Отиаи. Косинусный коэффициент — мера подобия между двумя массивами данных, вычисляемая как косинус угла между векторами в многомерном пространстве. В самом деле, двух пользователей разумно считать похожими, если угол между их векторами предпочтений мал.

Пусть даны два вектора признаков, 𝐴 и 𝐵, тогда косинусное сходство, 𝑐𝑜𝑠(𝜃), может быть представлено используя скалярное произведение и норму (Формула 2.6).

(2.6)

Косинусный коэффициент изменяется −1 ≤ 𝑐𝑜𝑠(𝜃) ≤ 1. Если 𝑐𝑜𝑠(𝜃) = 1 ∠θ = 0, то вкусы пользователей похожи. Если 𝑐𝑜𝑠(𝜃) = −1 ∠θ = 180, то вкусы пользователей противоположны. Если 𝑐𝑜𝑠(𝜃) = 0 ∠θ = 90 – зависимость между предпочтениями пользователей не просматривается.

Одна из причин популярности косинусного сходства состоит в том, что оно эффективно в качестве оценочной меры, особенно для разреженных векторов, так как необходимо учитывать только ненулевые измерения.

1. Коэффициент корреляции Пирсона.

Похожесть объектов 𝑖 и 𝑡 определяется с помощью корреляции Пирсона по Формуле 2.7.

(2.7)

где – множество пользователей, которые оценили объекты и ;

– оценка, поставленная пользователем объекту ;

– оценка, поставленная пользователем объекту ;

– средняя оценка пользователя .

Коэффициент корреляции Пирсона изменяется в интервале от — 1 до +1; безразмерен, т. е. не имеет единиц измерения; указывает, как близко расположены точки к прямой линии.

# 3 ДОКУМЕНТАЦИЯ К ДАННЫМ

## 3.1 Описание предметной области

Данные взяты из открытого сервиса MovieLens, предоставляемого исследовательской группой GroupLens Университета Миннесоты. Датасет ml-latest-small содержит информацию о рейтингах и тегах пользователей для фильмов.

Целью использования этих данных является построение рекомендательной системы фильмов, которая может предсказывать оценки для фильмов, не просмотренных пользователем, и рекомендовать новые фильмы на основе исторических оценок и тегов.

Датасет состоит из следующих сущностей:

* пользователи (User): анонимизированные идентификаторы пользователей, которые выставляли оценки фильмов и добавляли теги;
* фильмы (Movie): идентификаторы, название, жанры, год выпуска и ссылки на внешние источники (IMDb, TMDb);
* рейтинги (Rating): пользователь, фильм, оценка от 0.5 до 5 и временная метка;
* теги (Tag): пользователь, фильм, текст тега и временная метка.

## 3.2 Анализ данных

Датасет ml-latest-small представляет собой компактную выборку данных MovieLens, содержащую оценки пользователей и информацию о фильмах. Он включает чуть более ста тысяч рейтингов, чуть меньше десяти тысяч фильмов и около шести сотен пользователей, что делает его удобным для экспериментов и обучения рекомендательных систем без необходимости использования мощных вычислительных ресурсов.

Каждая запись о рейтинге состоит из идентификатора пользователя, идентификатора фильма, выставленной оценки и временной метки. Основная информация о данных в файле ratings.csv представлена на Рисунке 3.2.1.

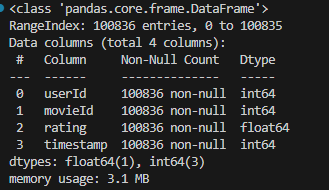


Рисунок 3.2.1 - Основная информация о данных в файле ratings.csv

Распределение выставленных оценок представлено на Рисунке 3.2.2.

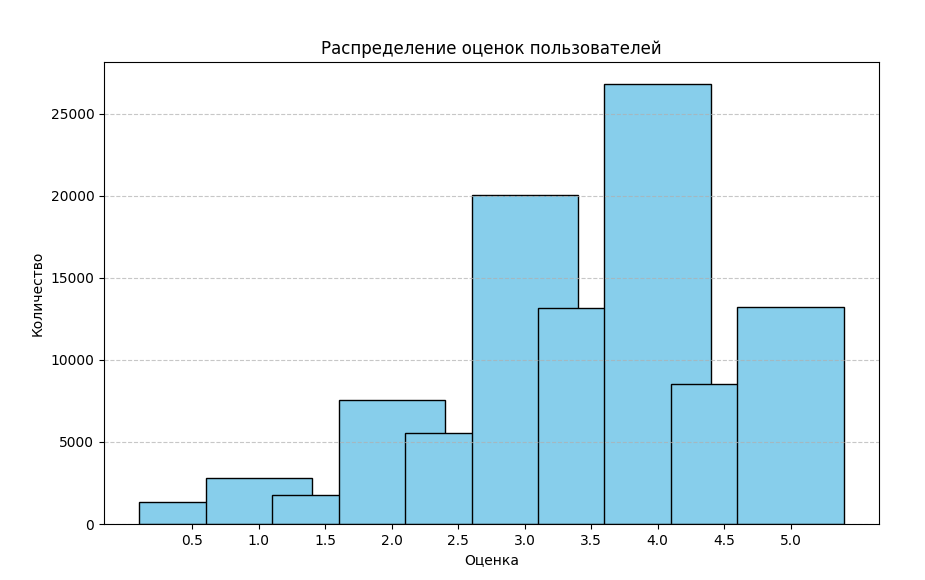


Рисунок 3.2.2 – Распределение оценок

Оценки распределены по шкале от 0.5 до 5 с шагом 0.5, что позволяет достаточно точно моделировать предпочтения пользователей, одновременно упрощая обработку данных. Анализ распределения оценок показывает, что наиболее популярные значения находятся в диапазоне от 3 до 4 баллов, что отражает склонность пользователей давать средние и положительные оценки. Более экстремальные оценки, например 0.5 или 5, встречаются реже, но именно они могут быть особенно информативны при построении рекомендательных моделей, так как ясно отражают сильные предпочтения или отторжение.

Фильмы в датасете сопровождаются информацией о жанрах. Основная информация о данных в файле movies.csv отображена на Рисунке 3.2.3.

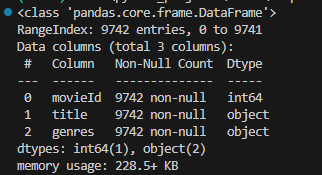


Рисунок 3.2.3 - Основная информация о данных в файле movies.csv

Датафрейм не содержит нулевых значений или пропусков. Первые 5 строк датафрейма представлены на Рисунке 3.2.4.

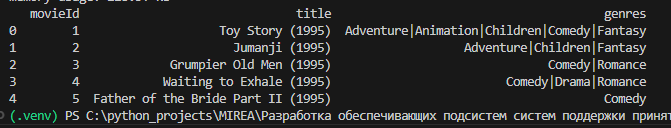


Рисунок 3.2.4 – Первые пять строк датафрейма movies

Каждый фильм может относиться к нескольким жанрам, которые объединены в одну строку через символ |. Анализ жанров показывает, что большинство фильмов имеют два-три жанра, что отражает реальную практику киноиндустрии, где произведения редко ограничиваются одним жанром. Также есть небольшое количество фильмов без указанных жанров, что требует дополнительной предобработки для корректного включения таких данных в рекомендательную систему.

В названиях фильмов часто содержится год выпуска, заключённый в скобки. Это позволяет извлечь дополнительную информацию о времени выхода фильма и анализировать данные по временным периодам.

Теги, оставленные пользователями, представляют собой текстовые описания, которые дают дополнительные сведения о предпочтениях. Информация о данных в файле tags.csv представлена на Рисунке 3.2.5.

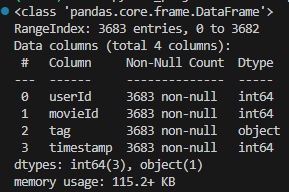


Рисунок 3.2.5 - Основная информация о данных в файле tags.csv

Анализ тегов показывает, что пользователи используют их довольно разнообразно: одни ограничиваются жанровыми метками, другие оставляют более субъективные описания вроде «funny» или «thriller». Теги помогают расширить рекомендации, особенно в гибридных системах, где учитываются как оценки, так и семантические характеристики фильмов.

Временные метки, присутствующие как в рейтингах, так и в тегах, позволяют анализировать динамику активности пользователей. С их помощью можно определить, когда пользователи наиболее активно выставляли оценки, выявить сезонные колебания интереса к фильмам или построить модели, учитывающие эволюцию предпочтений во времени.

Общий анализ показывает, что датасет ml-latest-small хорошо сбалансирован для экспериментов с рекомендательными системами. Он содержит достаточно данных для выявления закономерностей в поведении пользователей и предпочтениях фильмов, но при этом не требует огромных вычислительных ресурсов, что позволяет быстро проводить предобработку и обучение моделей.

## 3.3 Предобработка данных

В рамках предобработки удален год выпуска фильма из названия и добавлен отдельный столбец year в соответствующий датафрейм.

Временные метки приведены к типу данных datetime для более удобной обработки. Жанры преобразованы в список, обработаны фильмы с отсутствующими жанрами.

# 4 ПРАКТИЧЕСКАЯ ЧАСТЬ

## 4.1 Функциональные возможности

В практической части реализована интерактивная консольная система для работы с данными MovieLens. Основная цель заключалась в создании среды, позволяющей пользователю просматривать фильмы, искать их по названию и жанрам, оценивать и получать рекомендации на основе собственных оценок и оценок других пользователей. Полный код реализации кинотеатра с рекомендательной системой представлен в Приложении А.

Главное меню кинотеатра представлено на Рисунке 4.1.1.



Рисунок 4.1.1 – Главное меню

Пользователю доступен просмотр каталога фильмов, ленты рекомендаций, ранее поставленных оценок, также реализована возможность выхода из приложения.

Каталог фильмов представлен на Рисунке 4.1.2.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.1.2 – Каталог фильмов

Сортировка каталога возможна по году выпуска фильм, по жанрам, по идентификатору. Пример сортировки каталога представлен на Рисунке 4.1.3.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.1.3 – Сортировка каталога по году по убыванию

Добавлен механизм поиска по каталогу с фильтрацией по году выпуска, жанрам. Поиск осуществляется по названию путем проверки заданной подстроки. Интерфейс поиска представлен на Рисунке 4.1.4.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.1.4 – Интерфейс поиска

Результат поиска на запрос alien представлены на Рисунке 4.1.5.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.1.5 – Результаты поиска на запрос alien

Для составлений рекомендаций пользователю необходимо поставить оценки фильмам. Для этого реализована возможность открытия «страницы» фильма, где доступен просмотр тэгов фильма и присутствует возможность оценить фильм.

Интерфейс страницы фильма представлен на Рисунке 4.1.6.

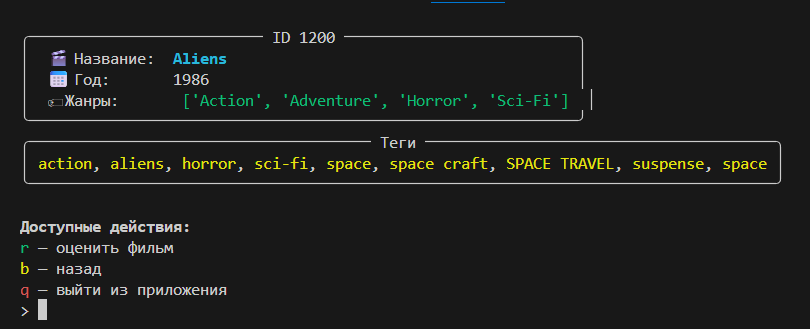


Рисунок 4.1.6 – Интерфейс страницы фильма

Пример выставления оценки показан на Рисунке 4.1.7.

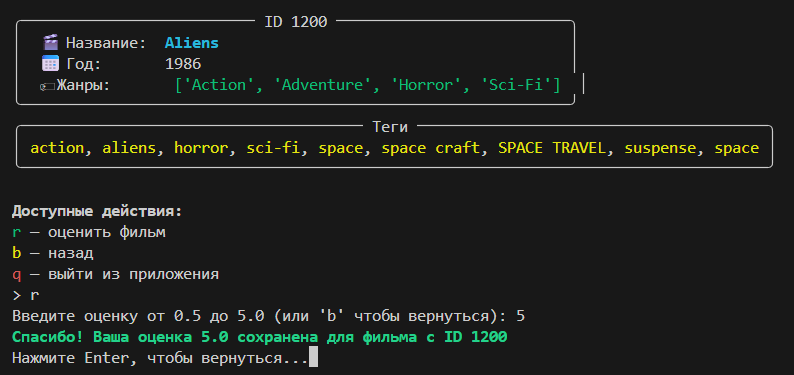


Рисунок 4.1.7 – Выставление оценки для фильма

Проставленные оценки можно посмотреть на отдельной странице в главном меню (Рисунок 4.1.8).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.1.8 – Проставленные оценки

Добавлена возможность удалить оценку по идентификатору фильма (Рисунок 4.1.9).

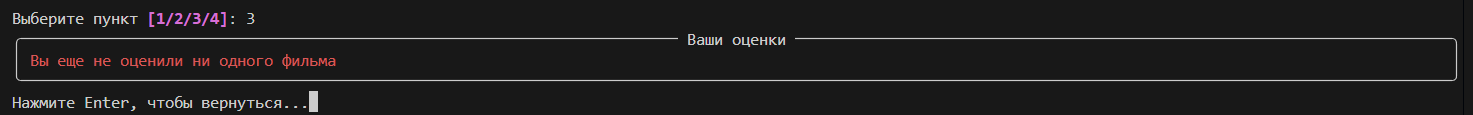


Рисунок 4.1.9 – Удаление оценки

Оценки на фильм с идентификатором 1200 успешно удалена, текущий пользователь не оценил ни одного фильма.

## 4.2 Рекомендательная система

Основные вычисления рекомендаций выполняются в методе show\_recommendations. После того, как пользователь оценил некоторое число фильмов для него составляется лента рекомендаций.

Реализована возможность выбора стратегии рекомендаций (User-based, Item-Based), а также метрики близости (Коэффициент Пирсона, Евклидова норма, расстояние Жаккара, косинусная мера). Пользователь может выбирать метрику и стратегию при каждом запросе рекомендаций, что позволяет сравнивать качество работы различных подходов.

Алгоритм User-based стратегии основан на поиске пользователей, чьи профили наиболее похожи на профиль текущего пользователя. Для каждой пары пользователей вычисляется мера сходства по выбранной метрике. После этого для каждого фильма, который пользователь не оценивал, предсказывается рейтинг по формуле среднего с взвешенными отклонениями (Формула 4.1).

(4.1)

где – средняя оценка, проставленная пользователем ;

– мера схожести пользователей и .

Для Item-based стратегии прогноз строится на основе сходства между фильмами. Для каждого фильма, который пользователь ещё не оценивал, ищутся другие фильмы, которые он уже оценил. На основе выбранной меры близости рассчитываются веса, и итоговый рейтинг оценивается по формуле взвешенного отклонения от среднего по фильмам (Формула 4.2).

(4.2)

где – средняя оценка, проставленная объекту ;

– мера схожести пользователей и .

Создан пользователь с оценками, представленными на Рисунке 4.1.10.

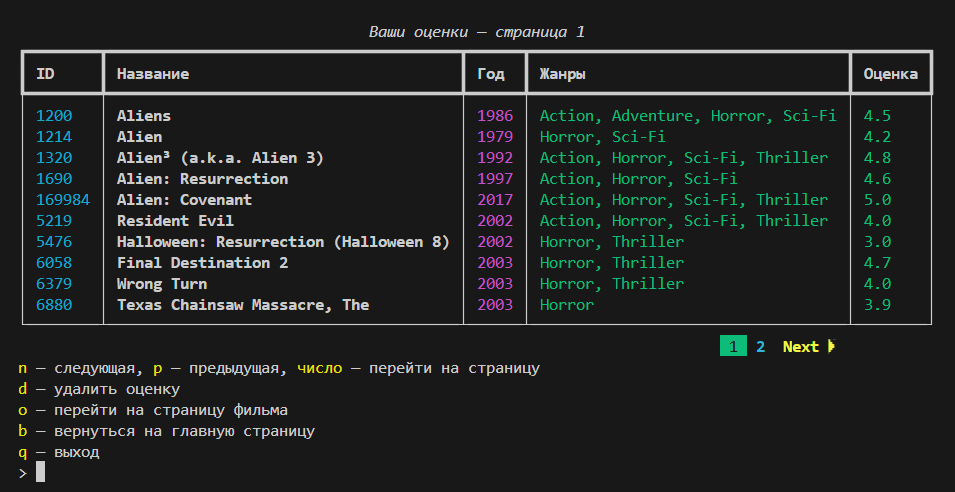


Рисунок 4.1.10 – Пользователь с заданными оценками

Преимущественно оценки поставлены фильмам ужасов.

Первая страница рекомендаций, основанных на подходе User-based изображена на Рисунке 4.1.11.

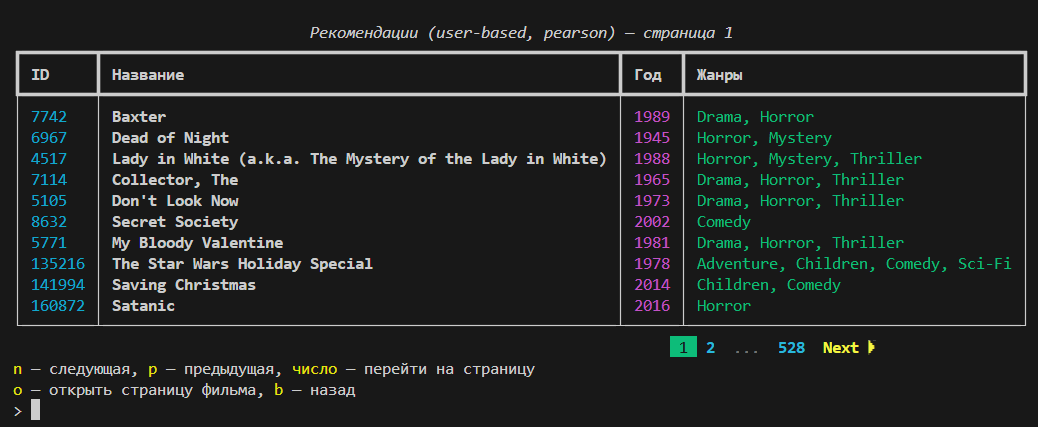


Рисунок 4.1.11 - Первая страница рекомендаций, основанных на подходе User-based

Рекомендованные фильмы преимущественно содержат жанр ужасы, как и ожидалось.

Для количественной оценки качества использованы метрики RMSE и MAE, которые рассчитываются на отложенной тестовой части данных (20 % случайно выбранных известных оценок заменяются на NaN, после чего восстанавливаются алгоритмом). Расчёт метрик вынесен в отдельный метод \_compute\_metrics\_for\_plot, а визуализация производится функцией plot\_similarity\_metrics, которая строит столбчатые диаграммы для каждой метрики и меры сходства (Рисунок 4.1.12).

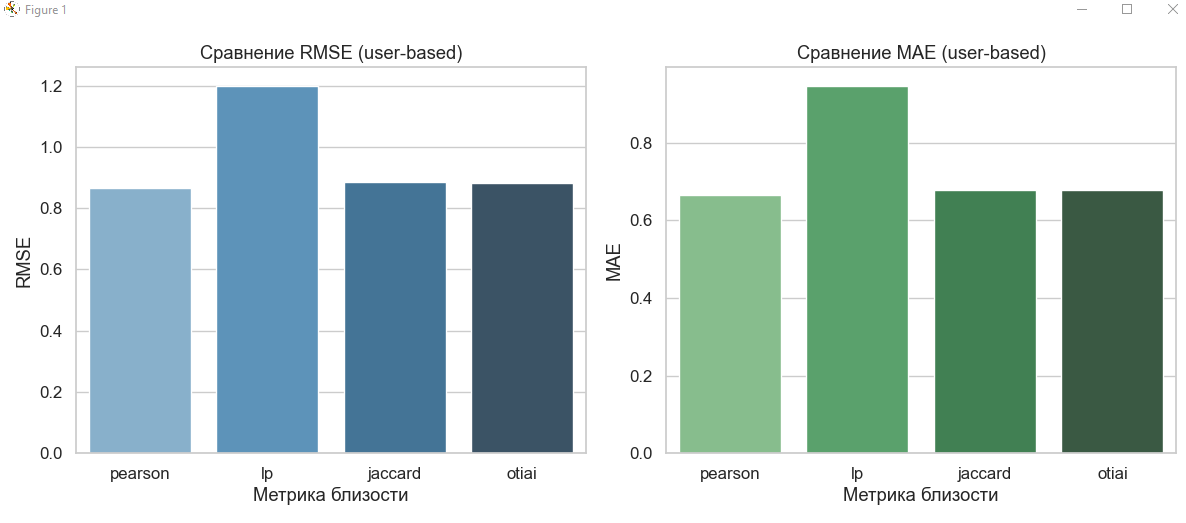


Рисунок 4.1.12 – Расчет метрик

Лучшей метрикой близости является коэффициент Пирсона, а худшая метрика – Евклидово расстояние.

# ЗАКЛЮЧЕНИЕ

В ходе данной практической работы успешно достигнута поставленная цель: приобретены навыки реализации и проведен сравнительный анализ анамнестических методов коллаборативной фильтрации для построения рекомендательной системы. Все поставленные задачи были выполнены в полном объеме.

Разработана функциональная консольная система на языке Python, интегрирующая в себя интерактивный каталог фильмов на основе датасета MovieLens и две ключевые стратегии рекомендаций: User-Based (на основе схожести пользователей) и Item-Based (на основе схожести предметов). В рамках этих стратегий реализованы и эмпирически сравнены четыре математические меры сходства: коэффициент корреляции Пирсона, Евклидова норма, расстояние Жаккара и косинусная мера.

Проведенное исследование и расчет метрик ошибок (RMSE и MAE) позволили сделать обоснованный вывод о том, что наилучшее качество прогноза для данной предметной области и набора данных обеспечивает коэффициент корреляции Пирсона. Это объясняется его способностью учитывать не только абсолютные значения оценок, но и относительные отклонения от среднего пользовательского рейтинга, что делает его более устойчивым к индивидуальным особенностям выставления оценок. В то же время Евклидова метрика показала наихудшие результаты, что может быть связано с ее чувствительностью к величине и масштабу данных.

# СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Сорокин, А. Б. Безусловная оптимизация. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин, О. В. Платонова, Л. М. Железняк — М. РТУ МИРЭА , 2020.
2. Сорокин, А. Б. Введение в генетические алгоритмы: теория, расчеты и приложения. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин — М. МИРЭА , 2018.
3. Рекомендательные системы: user-based и item-based [Электронный ресурс]: Habr. URL: https://habr.com/ru/companies/surfingbird/articles/139518/ (Дата обращения: 14.10.2025).
4. Рекомендательные системы [Электронный ресурс]: URL: https://neerc.ifmo.ru/wiki/index.php?title=Рекомендательные\_системы (Дата обращения: 12.10.2025).
5. MovieLens [Электронный ресурс]: URL: https://grouplens.org/datasets/movielens/ (Дата обращения: 13.10.2025).
6. Item-Based Collaborative Filtering Recommendation Algorithms [Электронный ресурс]: URL: https://files.grouplens.org/papers/www10\_sarwar.pdf (Дата обращения: 14.10.2025).

# ПРИЛОЖЕНИЯ

Приложение А — Реализация кинотеатра с встроенной рекомендательной системой.

### Приложение А

Реализация кинотеатра с встроенной рекомендательной системой

Листинг А – Реализация кинотеатра с встроенной рекомендательной системой

import os

import re

import pandas as pd

import numpy as np

from rich.console import Console

from rich.table import Table

from rich.prompt import Confirm, Prompt

from rich.panel import Panel

from rich.align import Align

from rich.text import Text

from enum import Enum

import seaborn as sns

import matplotlib.pyplot as plt

def extract\_year(title):

    title = title.strip()

    match = re.search(r"\((\d{4})\)\s\*$", title)

    if match:

        return int(match.group(1))

    return np.nan

class Genres(Enum):

    Action = "Action"

    Adventure = "Adventure"

    Animation = "Animation"

    Children = "Children's"

    Comedy = "Comedy"

    Crime = "Crime"

    Documentary = "Documentary"

    Drama = "Drama"

    Fantasy = "Fantasy"

    FilmNoir = "Film-Noir"

    Horror = "Horror"

    Musical = "Musical"

    Mystery = "Mystery"

    Romance = "Romance"

    SciFi = "Sci-Fi"

    Thriller = "Thriller"

    War = "War"

    Western = "Western"

class MovieLensCinema:

    def \_\_init\_\_(self, path, per\_page=10):

        self.path = path

        self.links = None

        self.movies = None

        self.ratings = None

        self.tags = None

        self.console = Console()

        self.rates = pd.DataFrame(columns=["movieId", "rating"]).set\_index("movieId")

        self.load\_data()

        self.per\_page = per\_page

    def load\_data(self):

Продолжение Листинга А

        links\_path = os.path.join(self.path, "links.csv")

        movies\_path = os.path.join(self.path, "movies.csv")

        ratings\_path = os.path.join(self.path, "ratings.csv")

        tags\_path = os.path.join(self.path, "tags.csv")

        if os.path.exists(links\_path):

            self.links = pd.read\_csv(

                links\_path,

                encoding="utf-8",

                index\_col="movieId",

                dtype={"imdbId": "int64", "tmdbId": "Int64"},

            )

        else:

            raise FileNotFoundError("Не найдено файла links.csv")

        if os.path.exists(movies\_path):

            self.movies = pd.read\_csv(

                movies\_path, encoding="utf-8", index\_col="movieId", quotechar='"'

            )

            self.movies["title"] = self.movies["title"].str.strip()

            self.movies["genres"] = self.movies["genres"].apply(

                lambda x: [] if x == "(no genres listed)" else x.split("|")

            )

            self.movies["year"] = self.movies["title"].apply(extract\_year)

            self.movies["year"] = self.movies["year"].astype("Int64")

            self.movies["title"] = self.movies["title"].apply(

                lambda t: re.sub(r"\s\*\(\d{4}\)$", "", t)

            )

        else:

            raise FileNotFoundError("Не найдено файла movies.csv")

        if os.path.exists(ratings\_path):

            self.ratings = pd.read\_csv(

                ratings\_path,

                encoding="utf-8",

                dtype={

                    "userId": "int32",

                    "movieId": "int32",

                    "rating": "float32",

                    "timestamp": "int64",

                },

            )

            self.ratings["datetime"] = pd.to\_datetime(

                self.ratings["timestamp"], unit="s"

            )

            self.ratings = self.ratings.drop(columns=["timestamp"])

            self.ratings.set\_index(["userId", "movieId"], inplace=True)

        else:

            raise FileNotFoundError("Не найдено файла ratings.csv")

        if os.path.exists(tags\_path):

            self.tags = pd.read\_csv(

                tags\_path,

                dtype={

                    "userId": "int32",

                    "movieId": "int32",

                    "tag": "string",

                    "timestamp": "int64",

                },

Продолжение Листинга А

                quotechar='"',

            )

            self.tags["datetime"] = pd.to\_datetime(self.tags["timestamp"], unit="s")

            self.tags.set\_index(["userId", "movieId"], inplace=True)

        else:

            raise FileNotFoundError("Не найдено файла tags.csv")

    def get\_rating\_matrix(self):

        df = self.ratings.reset\_index()

        rating\_matrix = df.pivot\_table(

            index="userId", columns="movieId", values="rating"

        )

        return rating\_matrix

    def show\_movies(self, page=1):

        start = (page - 1) \* self.per\_page

        end = start + self.per\_page

        subset = self.movies.iloc[start:end]

        table = Table(title=f"🎬 Список фильмов — страница {page}")

        table.add\_column("ID", style="cyan", no\_wrap=True)

        table.add\_column("Название", style="bold")

        table.add\_column("Год", justify="center", style="magenta")

        table.add\_column("Жанры", style="green")

        for movie\_id, row in subset.iterrows():

            genres\_str = ", ".join(row["genres"]) if row["genres"] else "—"

            year\_str = str(row["year"]) if not pd.isna(row["year"]) else "—"

            table.add\_row(str(movie\_id), row["title"], year\_str, genres\_str)

        self.console.print(table)

    def run(self):

        page = 1

        total\_pages = (len(self.movies) - 1) // self.per\_page + 1

        while True:

            os.system("cls" if os.name == "nt" else "clear")

            self.show\_movies(page=page)

            self.show\_paginator(page, total\_pages)

            self.console.print(

                "[yellow]n[/yellow] — следующая, [yellow]p[/yellow] — предыдущая, "

                "[yellow]число[/yellow] — перейти на страницу\n"

                "[yellow]f[/yellow] — поиск\n"

                "[yellow]s[/yellow] — отсортировать результаты\n"

                "[yellow]o[/yellow] — перейти на страницу фильма\n"

                "[yellow]b[/yellow] — вернуться на главную страницу\n"

                "[yellow]q[/yellow] — выход"

            )

            choice = input("> ").strip().lower()

            if choice == "n" and page < total\_pages:

                page += 1

            elif choice == "p" and page > 1:

Продолжение Листинга А

                page -= 1

            elif choice.isdigit():

                num = int(choice)

                if 1 <= num <= total\_pages:

                    page = num

            elif choice == "f":

                self.search\_movies()

            elif choice == "s":

                self.sort\_movies(self.movies)

            elif choice == "o":

                movie\_id\_str = Prompt.ask("Введите ID фильма (или 'b' для отмены)")

                if movie\_id\_str.lower() == "b":

                    continue

                if movie\_id\_str.isdigit():

                    self.show\_movie\_page(int(movie\_id\_str))

                else:

                    self.console.print("[red]Некорректный ID[/red]")

                    input("Нажмите Enter, чтобы вернуться...")

            elif choice == "b":

                break

            elif choice == "q":

                if Confirm.ask("Are you sure?"):

                    exit(0)

    def menu(self):

        while True:

            os.system("cls" if os.name == "nt" else "clear")

            self.console.print(

                Panel.fit(

                    "[bold magenta]🎬 Добро пожаловать в консольный кинотеатр![/bold magenta]\nВыберите действие:"

                ),

                justify="center",

            )

            self.console.print("[cyan]1.[/cyan] 📽 Подборка фильмов")

            self.console.print("[cyan]2.[/cyan] 🔍 Мои рекомендации")

            self.console.print("[cyan]3.[/cyan] 🔍 Мои оценки")

            self.console.print("[cyan]4.[/cyan] ❌ Выход")

            choice = Prompt.ask("\nВыберите пункт", choices=["1", "2", "3", "4"])

            if choice == "1":

                self.run()

            elif choice == "2":

                self.show\_recommendations()

            elif choice == "3":

                self.show\_my\_ratings()

            elif choice == "4":

                if Confirm.ask("[bold red]Вы действительно хотите выйти?[/bold red]"):

                    break

    def show\_movie\_page(self, movie\_id: int):

        """Страница фильма с информацией, тегами и возможностью оценки"""

        if movie\_id not in self.movies.index:

            self.console.print(f"[red]Фильм с ID {movie\_id} не найден.[/red]")

Продолжение Листинга А

            input("Нажмите Enter, чтобы вернуться...")

            return

        while True:

            os.system("cls" if os.name == "nt" else "clear")

            movie = self.movies.loc[movie\_id]

            info\_table = Table(show\_header=False, box=None)

            info\_table.add\_row(

                "🎬 Название:", f"[bold cyan]{movie['title']}[/bold cyan]"

            )

            info\_table.add\_row(

                "📅 Год:", str(movie["year"]) if movie["year"] == movie["year"] else "—"

            )

            info\_table.add\_row("🏷 Жанры:", f"[green]{movie['genres']}[/green]")

            self.console.print(Panel(info\_table, title=f"ID {movie\_id}", expand=False))

            if self.tags is not None:

                try:

                    df = self.tags.xs(movie\_id, level="movieId")

                    movie\_tags = df["tag"].tolist()

                except KeyError:

                    movie\_tags = []

                if movie\_tags:

                    tags\_str = ", ".join(f"[yellow]{t}[/yellow]" for t in movie\_tags)

                    self.console.print(Panel(tags\_str, title="Теги", expand=False))

                else:

                    self.console.print(

                        Panel(

                            "[italic grey]Для этого фильма нет тегов[/italic grey]",

                            title="Теги",

                            expand=False,

                        )

                    )

            self.console.print(

                "\n[bold]Доступные действия:[/bold]\n"

                "[green]r[/green] — оценить фильм\n"

                "[yellow]b[/yellow] — назад\n"

                "[red]q[/red] — выйти из приложения"

            )

            choice = input("> ").strip().lower()

            if choice == "r":

                self.rate\_movie(movie\_id)

            elif choice == "b":

                break

            elif choice == "q":

                if Confirm.ask("Are you sure?"):

                    exit(0)

    def rate\_movie(self, movie\_id: int):

        """Простейшая система выставления оценки"""

        while True:

Продолжение Листинга А

            rating = Prompt.ask(

                "Введите оценку от 0.5 до 5.0 (или 'b' чтобы вернуться)"

            )

            if rating.lower() == "b":

                break

            try:

                rating = float(rating)

                if 0.5 <= rating <= 5.0:

                    self.rates.loc[movie\_id, "rating"] = rating

                    self.console.print(

                        f"[bold green]Спасибо! Ваша оценка {rating} сохранена для фильма c ID {movie\_id}[/bold green]"

                    )

                    input("Нажмите Enter, чтобы вернуться...")

                    break

                else:

                    self.console.print(

                        "[red]Оценка должна быть в диапазоне 0.5–5.0[/red]"

                    )

                    input("Нажмите Enter, чтобы вернуться...")

                    break

            except ValueError:

                self.console.print("[red]Введите корректное число или 'b'[/red]")

                input("Нажмите Enter, чтобы вернуться...")

                break

    def search\_movies(self):

        """Поиск фильмов по названию с фильтрацией по году и жанрам"""

        os.system("cls" if os.name == "nt" else "clear")

        search\_query = (

            Prompt.ask("Введите название фильма (Enter — пропустить)").strip().lower()

        )

        selected\_genres = self.choose\_genres()

        year\_op, year\_val = self.choose\_year\_filter()

        def apply\_filters():

            df = self.movies

            if search\_query:

                df = df[df["title"].str.lower().str.contains(search\_query, na=False)]

            if selected\_genres:

                df = df[

                    df["genres"].apply(

                        lambda g: all(gen in g for gen in selected\_genres)

                    )

                ]

            if year\_op is not None and year\_val is not None:

                if year\_op == "=":

                    df = df[df["year"] == year\_val]

                elif year\_op == ">":

                    df = df[df["year"] > year\_val]

                elif year\_op == "<":

                    df = df[df["year"] < year\_val]

            return df

        filtered\_df = apply\_filters()

        if filtered\_df.empty:

            self.console.print(

                Panel("[red]❌ Фильмы не найдены[/red]", title="Результат")

Продолжение Листинга А

            )

            input("Нажмите Enter, чтобы вернуться...")

            return

        page = 1

        per\_page = self.per\_page

        while True:

            os.system("cls" if os.name == "nt" else "clear")

            total\_pages = max(1, (len(filtered\_df) - 1) // per\_page + 1)

            page = min(page, total\_pages)

            start = (page - 1) \* per\_page

            end = start + per\_page

            page\_data = filtered\_df.iloc[start:end]

            table = Table(

                title=f"Результаты поиска: {len(filtered\_df)}",

                show\_header=True,

                header\_style="bold magenta",

            )

            table.add\_column("ID", style="cyan", width=6)

            table.add\_column("Название", style="white")

            table.add\_column("Год", style="yellow", width=8)

            table.add\_column("Жанры", style="green")

            for idx, row in page\_data.iterrows():

                year = str(row["year"]) if not pd.isna(row["year"]) else "—"

                genres = ", ".join(row["genres"]) if row["genres"] else "—"

                table.add\_row(str(idx), row["title"], year, genres)

            self.console.print(table)

            self.show\_paginator(page, total\_pages)

            status = f"[bold]Фильтры:[/bold] Название: [cyan]{search\_query or '—'}[/cyan], Жанры: [cyan]{', '.join(selected\_genres) if selected\_genres else '—'}[/cyan], Год: [cyan]{year\_op + str(year\_val) if year\_op and year\_val else '—'}[/cyan]"

            self.console.print(status)

            self.console.print(

                "[yellow]n[/yellow] — следующая, [yellow]p[/yellow] — предыдущая, [yellow]число[/yellow] — перейти на страницу\n"

                "[yellow]s[/yellow] — отсортировать результаты\n"

                "[yellow]o[/yellow] — открыть страницу фильма по ID\n"

                "[yellow]r[/yellow] — сброс фильтров\n"

                "[yellow]b[/yellow] — назад"

            )

            choice = input("> ").strip().lower()

            if choice == "n" and page < total\_pages:

                page += 1

            elif choice == "p" and page > 1:

                page -= 1

            elif choice == "s":

                self.sort\_movies(filtered\_df)

            elif choice.isdigit():

                num = int(choice)

                if 1 <= num <= total\_pages:

Продолжение Листинга А

                    page = num

            elif choice == "o":

                movie\_id\_str = Prompt.ask("Введите ID фильма (или 'b' чтобы вернуться)")

                if movie\_id\_str.lower() == "b":

                    continue

                try:

                    movie\_id = int(movie\_id\_str)

                    if movie\_id in self.movies.index:

                        self.show\_movie\_page(movie\_id)

                    else:

                        self.console.print("[red]Фильм с таким ID не найден[/red]")

                        input("Нажмите Enter, чтобы продолжить...")

                except ValueError:

                    self.console.print("[red]Введите корректный ID[/red]")

                    input("Нажмите Enter, чтобы продолжить...")

            elif choice == "r":

                search\_query = ""

                selected\_genres = []

                year\_op, year\_val = None, None

                filtered\_df = self.movies

                page = 1

            elif choice == "b":

                break

    def choose\_genres(self):

        """Выбор одного или нескольких жанров через консоль с таблицей"""

        genre\_list = list(Genres)

        table = Table(

            title="Выберите жанры", show\_header=True, header\_style="bold magenta"

        )

        table.add\_column("№", justify="center", style="cyan", width=4)

        table.add\_column("Жанр", justify="left", style="green")

        for i, genre in enumerate(genre\_list, 1):

            table.add\_row(str(i), genre.value)

        self.console.print(table)

        choice = Prompt.ask(

            "Введите номера жанров через пробел (Enter — пропустить)"

        ).strip()

        if not choice:

            return []

        selected\_genres = []

        for num in choice.split():

            num = num.strip()

            if num.isdigit():

                idx = int(num) - 1

                if 0 <= idx < len(genre\_list):

                    selected\_genres.append(genre\_list[idx].value)

        return selected\_genres

    def choose\_year\_filter(self):

        """Выбор фильтра по году с оператором"""

        op = Prompt.ask(

            "Выберите оператор для фильтра по году",

            choices=[">", "<", "="],

            default="=",

Продолжение Листинга А

        )

        year\_str = Prompt.ask("Введите год").strip()

        if not year\_str.isdigit():

            self.console.print("[red]Некорректный год, фильтр не будет применён[/red]")

            return None, None

        return op, int(year\_str)

    def show\_paginator(self, page, total\_pages):

        """Красивый центрированный пагинатор"""

        paginator\_text = Text()

        last\_was\_ellipsis = False

        if page > 1:

            paginator\_text.append("◀ Prev ", style="bold yellow")

        else:

            paginator\_text.append("         ")

        for p in range(1, total\_pages + 1):

            if p == 1 or p == total\_pages or abs(p - page) <= 1:

                if p == page:

                    paginator\_text.append(f" {p} ", style="reverse green")

                else:

                    paginator\_text.append(f" {p} ", style="bold cyan")

                last\_was\_ellipsis = False

            else:

                if not last\_was\_ellipsis:

                    paginator\_text.append(" ... ", style="dim")

                    last\_was\_ellipsis = True

        if page < total\_pages:

            paginator\_text.append(" Next ▶", style="bold yellow")

        self.console.print(Align.center(paginator\_text))

    def show\_my\_ratings(self):

        if self.rates.empty:

            self.console.print(

                Panel(

                    "[red]Вы еще не оценили ни одного фильма[/red]", title="Ваши оценки"

                )

            )

            input("Нажмите Enter, чтобы вернуться...")

            return

        page = 1

        df = self.rates.merge(self.movies, on="movieId")

        total\_pages = (len(df) - 1) // self.per\_page + 1

        while len(df):

            os.system("cls" if os.name == "nt" else "clear")

            start = (page - 1) \* self.per\_page

            end = start + self.per\_page

            subset = df.iloc[start:end]

            rates = Table(show\_header=True, title=f"Ваши оценки — страница {page}")

            rates.add\_column("ID", style="cyan", no\_wrap=True)

            rates.add\_column("Название", style="bold")

Продолжение Листинга А

            rates.add\_column("Год", justify="center", style="magenta")

            rates.add\_column("Жанры", style="green")

            rates.add\_column("Оценка", style="green")

            for movie\_id, row in subset.iterrows():

                genres\_str = ", ".join(row["genres"]) if row["genres"] else "—"

                year\_str = str(row["year"]) if not pd.isna(row["year"]) else "—"

                rates.add\_row(

                    str(movie\_id),

                    row["title"],

                    year\_str,

                    genres\_str,

                    str(row["rating"]),

                )

            self.console.print(rates)

            self.show\_paginator(page, total\_pages)

            self.console.print(

                "[yellow]n[/yellow] — следующая, [yellow]p[/yellow] — предыдущая, "

                "[yellow]число[/yellow] — перейти на страницу\n"

                "[yellow]d[/yellow] — удалить оценку\n"

                "[yellow]o[/yellow] — перейти на страницу фильма\n"

                "[yellow]b[/yellow] — вернуться на главную страницу\n"

                "[yellow]q[/yellow] — выход"

            )

            choice = input("> ").strip().lower()

            if choice == "n" and page < total\_pages:

                page += 1

            elif choice == "p" and page > 1:

                page -= 1

            elif choice.isdigit():

                num = int(choice)

                if 1 <= num <= total\_pages:

                    page = num

            elif choice == "d":

                movie\_id\_str = Prompt.ask(

                    "Введите ID фильма, у которого хотите удалить оценку (или 'b' для отмены)"

                )

                if movie\_id\_str.lower() == "b":

                    continue

                if movie\_id\_str.isdigit():

                    id = int(movie\_id\_str)

                    if id in self.rates.index:

                        self.rates = self.rates.drop(index=id)

                        df = df.drop(index=id)

                    else:

                        self.console.print(

                            "[red]Не найдено оценки для фильма с заданным ID[/red]"

                        )

                        input("Нажмите Enter, чтобы вернуться...")

                else:

                    self.console.print("[red]Некорректный ID[/red]")

                    input("Нажмите Enter, чтобы вернуться...")

            elif choice == "o":

                movie\_id\_str = Prompt.ask("Введите ID фильма (или 'b' для отмены)")

Продолжение Листинга А

                if movie\_id\_str.lower() == "b":

                    continue

                if movie\_id\_str.isdigit():

                    self.show\_movie\_page(int(movie\_id\_str))

                else:

                    self.console.print("[red]Некорректный ID[/red]")

                    input("Нажмите Enter, чтобы вернуться...")

            elif choice == "b":

                break

            elif choice == "q":

                if Confirm.ask("Are you sure?"):

                    exit(0)

    def sort\_movies(self, frame):

        """Сортировка фильмов по различным полям"""

        sort\_fields = [

            ("ID", "movieId"),

            ("Название", "title"),

            ("Год", "year"),

            ("Жанры", "genres"),

        ]

        table = Table(

            title="Выберите поле для сортировки",

            show\_header=True,

            header\_style="bold magenta",

        )

        table.add\_column("№", justify="center", style="cyan", width=4)

        table.add\_column("Поле", style="green")

        for i, (label, \_) in enumerate(sort\_fields, 1):

            table.add\_row(str(i), label)

        self.console.print(table)

        choice = Prompt.ask("Введите номер поля (или Enter для отмены)").strip()

        if not choice.isdigit():

            return

        choice\_num = int(choice)

        if not 1 <= choice\_num <= len(sort\_fields):

            self.console.print("[red]Некорректный выбор[/red]")

            input("Нажмите Enter, чтобы вернуться...")

            return

        field\_label, field\_name = sort\_fields[choice\_num - 1]

        direction = Prompt.ask(

            "Выберите направление сортировки", choices=["asc", "desc"], default="asc"

        )

        ascending = direction == "asc"

        if field\_name == "movieId":

            sorted\_df = frame.sort\_index(ascending=ascending)

        elif field\_name == "genres":

            sorted\_df = frame.copy()

            sorted\_df["\_\_sort\_genre"] = sorted\_df["genres"].apply(

                lambda g: g[0] if g else ""

Продолжение Листинга А

            )

            sorted\_df = sorted\_df.sort\_values("\_\_sort\_genre", ascending=ascending).drop(

                columns="\_\_sort\_genre"

            )

        else:

            sorted\_df = frame.sort\_values(field\_name, ascending=ascending)

        self.paginated\_view(

            sorted\_df, title=f"Сортировка по {field\_label} ({direction})"

        )

    def paginated\_view(self, df, title="Список фильмов"):

        """Универсальный постраничный просмотр DataFrame фильмов"""

        page = 1

        total\_pages = max(1, (len(df) - 1) // self.per\_page + 1)

        while True:

            os.system("cls" if os.name == "nt" else "clear")

            start = (page - 1) \* self.per\_page

            end = start + self.per\_page

            subset = df.iloc[start:end]

            table = Table(title=f"{title} — страница {page}", show\_header=True)

            table.add\_column("ID", style="cyan", no\_wrap=True)

            table.add\_column("Название", style="bold")

            table.add\_column("Год", justify="center", style="magenta")

            table.add\_column("Жанры", style="green")

            for movie\_id, row in subset.iterrows():

                genres\_str = ", ".join(row["genres"]) if row["genres"] else "—"

                year\_str = str(row["year"]) if not pd.isna(row["year"]) else "—"

                table.add\_row(str(movie\_id), row["title"], year\_str, genres\_str)

            self.console.print(table)

            self.show\_paginator(page, total\_pages)

            self.console.print(

                "[yellow]n[/yellow] — следующая, [yellow]p[/yellow] — предыдущая, "

                "[yellow]число[/yellow] — перейти на страницу\n"

                "[yellow]o[/yellow] — открыть страницу фильма, [yellow]b[/yellow] — назад"

            )

            choice = input("> ").strip().lower()

            if choice == "n" and page < total\_pages:

                page += 1

            elif choice == "p" and page > 1:

                page -= 1

            elif choice.isdigit():

                num = int(choice)

                if 1 <= num <= total\_pages:

                    page = num

            elif choice == "o":

                movie\_id\_str = Prompt.ask("Введите ID фильма (или 'b' для отмены)")

                if movie\_id\_str.lower() == "b":

                    continue

                if movie\_id\_str.isdigit():

                    self.show\_movie\_page(int(movie\_id\_str))

            elif choice == "b":

Продолжение Листинга А

                break

        os.system("cls" if os.name == "nt" else "clear")

    def show\_recommendations(self):

        """Вывод рекомендаций для текущего пользователя, включая оценки из self.rates"""

        os.system("cls" if os.name == "nt" else "clear")

        R = self.get\_rating\_matrix()

        user\_ids = R.index.tolist()

        movie\_ids = R.columns.tolist()

        current\_user\_id = -1

        current\_user\_ratings = pd.Series(

            [np.nan] \* len(movie\_ids), index=movie\_ids, dtype=float

        )

        for movie\_id in self.rates.index:

            if movie\_id in movie\_ids:

                current\_user\_ratings[movie\_id] = self.rates.loc[movie\_id, "rating"]

        current\_user\_df = pd.DataFrame([current\_user\_ratings], index=[current\_user\_id])

        if current\_user\_df.notna().any().any():

            R = pd.concat([R, current\_user\_df])

            user\_ids.append(current\_user\_id)

            current\_user\_idx = R.index.get\_loc(current\_user\_id)

        else:

            self.console.print(

                Panel(

                    "[red]У вас нет оценок для рекомендаций. Сначала оцените несколько фильмов.[/red]",

                    title="Рекомендации",

                )

            )

            input("Нажмите Enter, чтобы вернуться...")

            return

        strategy = Prompt.ask(

            "Выберите стратегию [user-based/item-based]",

            choices=["user-based", "item-based"],

            default="user-based",

        )

        metric = Prompt.ask(

            "Выберите меру сходства [jaccard/lp/otiai/pearson]",

            choices=["jaccard", "lp", "otiai", "pearson"],

            default="pearson",

        )

        R\_values = R.values.astype(np.float64)

        n\_users, n\_items = R\_values.shape

        user\_means = np.nanmean(R\_values, axis=1)

        item\_means = np.nanmean(R\_values, axis=0)

        preds = np.full(n\_items, np.nan)

        if strategy == "user-based":

            sims = np.zeros(n\_users)

Продолжение Листинга А

            current\_ratings = R\_values[current\_user\_idx, :]

            for u in range(n\_users):

                if u == current\_user\_idx:

                    continue

                other\_ratings = R\_values[u, :]

                mask = ~np.isnan(current\_ratings) & ~np.isnan(other\_ratings)

                if np.sum(mask) == 0:

                    sims[u] = 0

                    continue

                if metric == "pearson":

                    sims[u] = np.corrcoef(current\_ratings[mask], other\_ratings[mask])[

                        0, 1

                    ]

                elif metric == "lp":

                    sims[u] = -np.linalg.norm(

                        current\_ratings[mask] - other\_ratings[mask], ord=2

                    )

                elif metric == "jaccard":

                    sims[u] = np.sum(

                        (current\_ratings[mask] > 0) & (other\_ratings[mask] > 0)

                    ) / np.sum((current\_ratings[mask] > 0) | (other\_ratings[mask] > 0))

                elif metric == "otiai":

                    sims[u] = np.sum(current\_ratings[mask] \* other\_ratings[mask]) / (

                        np.linalg.norm(current\_ratings[mask])

                        \* np.linalg.norm(other\_ratings[mask])

                    )

                else:

                    sims[u] = 0

            for i in range(n\_items):

                if not np.isnan(R\_values[current\_user\_idx, i]):

                    continue

                mask = ~np.isnan(R\_values[:, i])

                if np.sum(mask) == 0:

                    continue

                numerator = np.sum(sims[mask] \* (R\_values[mask, i] - user\_means[mask]))

                denominator = np.sum(np.abs(sims[mask])) + 1e-8

                preds[i] = user\_means[current\_user\_idx] + numerator / denominator

        else:

            for i in range(n\_items):

                if not np.isnan(R\_values[current\_user\_idx, i]):

                    continue

                rated\_mask = ~np.isnan(R\_values[current\_user\_idx, :])

                sims\_i = []

                ratings\_i = []

                for j in np.where(rated\_mask)[0]:

                    mask = ~np.isnan(R\_values[:, i]) & ~np.isnan(R\_values[:, j])

                    if np.sum(mask) == 0:

                        sim = 0

                    else:

                        if metric == "pearson":

                            sim = np.corrcoef(R\_values[mask, i], R\_values[mask, j])[

                                0, 1

                            ]

Продолжение Листинга А

                        elif metric == "lp":

                            sim = -np.linalg.norm(

                                R\_values[mask, i] - R\_values[mask, j], ord=2

                            )

                        elif metric == "jaccard":

                            sim = np.sum(

                                (R\_values[mask, i] > 0) & (R\_values[mask, j] > 0)

                            ) / np.sum(

                                (R\_values[mask, i] > 0) | (R\_values[mask, j] > 0)

                            )

                        elif metric == "otiai":

                            sim = np.sum(R\_values[mask, i] \* R\_values[mask, j]) / (

                                np.linalg.norm(R\_values[mask, i])

                                \* np.linalg.norm(R\_values[mask, j])

                            )

                        else:

                            sim = 0

                    sims\_i.append(sim)

                    ratings\_i.append(R\_values[current\_user\_idx, j] - item\_means[j])

                sims\_i = np.array(sims\_i)

                ratings\_i = np.array(ratings\_i)

                if np.sum(np.abs(sims\_i)) > 0:

                    preds[i] = item\_means[i] + np.dot(sims\_i, ratings\_i) / np.sum(

                        np.abs(sims\_i)

                    )

        recs\_df = pd.DataFrame({"movieId": movie\_ids, "pred\_rating": preds})

        recs\_df = recs\_df.dropna().sort\_values("pred\_rating", ascending=False)

        self.paginated\_view(

            df=self.movies.loc[recs\_df["movieId"]],

            title=f"Рекомендации ({strategy}, {metric})",

        )

    def plot\_similarity\_metrics(

        self, strategy: str = "user-based", test\_ratio: float = 0.2

    ):

        metrics = ["pearson", "lp", "jaccard", "otiai"]

        results = []

        for metric in metrics:

            rmse, mae = self.\_compute\_metrics\_for\_plot(metric, strategy, test\_ratio)

            results.append({"metric": metric, "RMSE": rmse, "MAE": mae})

        df = pd.DataFrame(results)

        sns.set\_theme(style="whitegrid", font\_scale=1.1)

        fig, axes = plt.subplots(1, 2, figsize=(12, 5))

        sns.barplot(x="metric", y="RMSE", data=df, ax=axes[0], palette="Blues\_d")

        axes[0].set\_title(f"Сравнение RMSE ({strategy})")

        axes[0].set\_xlabel("Метрика близости")

        axes[0].set\_ylabel("RMSE")

Продолжение Листинга А

        sns.barplot(x="metric", y="MAE", data=df, ax=axes[1], palette="Greens\_d")

        axes[1].set\_title(f"Сравнение MAE ({strategy})")

        axes[1].set\_xlabel("Метрика близости")

        axes[1].set\_ylabel("MAE")

        plt.tight\_layout()

        plt.show()

    def \_compute\_metrics\_for\_plot(self, metric: str, strategy: str, test\_ratio: float):

        R = self.get\_rating\_matrix().copy().values.astype(float)

        n\_users, n\_items = R.shape

        rng = np.random.default\_rng(42)

        train = R.copy()

        test\_mask = ~np.isnan(R) & (rng.random(R.shape) < test\_ratio)

        test\_true = np.full\_like(R, np.nan)

        test\_true[test\_mask] = R[test\_mask]

        train[test\_mask] = np.nan

        user\_means = np.nanmean(train, axis=1)

        item\_means = np.nanmean(train, axis=0)

        preds = np.full\_like(R, np.nan)

        for u in range(n\_users):

            if strategy == "user-based":

                sims = np.zeros(n\_users)

                current = train[u, :]

                for v in range(n\_users):

                    if v == u:

                        continue

                    other = train[v, :]

                    mask = ~np.isnan(current) & ~np.isnan(other)

                    if np.sum(mask) == 0:

                        sims[v] = 0

                        continue

                    if metric == "pearson":

                        sims[v] = np.corrcoef(current[mask], other[mask])[0, 1]

                    elif metric == "lp":

                        sims[v] = -np.linalg.norm(current[mask] - other[mask])

                    elif metric == "jaccard":

                        sims[v] = np.sum(

                            (current[mask] > 0) & (other[mask] > 0)

                        ) / np.sum((current[mask] > 0) | (other[mask] > 0))

                    elif metric == "otiai":

                        sims[v] = np.sum(current[mask] \* other[mask]) / (

                            np.linalg.norm(current[mask]) \* np.linalg.norm(other[mask])

                        )

                    else:

                        sims[v] = 0

                for i in range(n\_items):

                    if not np.isnan(train[u, i]):

                        continue

                    mask = ~np.isnan(train[:, i])

                    if np.sum(mask) == 0:

                        continue

                    numerator = np.sum(sims[mask] \* (train[mask, i] –

user\_means[mask]))

Окончание Листинга А

                    denominator = np.sum(np.abs(sims[mask])) + 1e-8

                    preds[u, i] = user\_means[u] + numerator / denominator

            else:

                for i in range(n\_items):

                    if not np.isnan(train[u, i]):

                        continue

                    rated\_mask = ~np.isnan(train[u, :])

                    sims\_i = []

                    ratings\_i = []

                    for j in np.where(rated\_mask)[0]:

                        mask = ~np.isnan(train[:, i]) & ~np.isnan(train[:, j])

                        if np.sum(mask) == 0:

                            sim = 0

                        else:

                            if metric == "pearson":

                                sim = np.corrcoef(train[mask, i], train[mask, j])[0, 1]

                            elif metric == "lp":

                                sim = -np.linalg.norm(train[mask, i] - train[mask, j])

                            elif metric == "jaccard":

                                sim = np.sum(

                                    (train[mask, i] > 0) & (train[mask, j] > 0)

                                ) / np.sum((train[mask, i] > 0) | (train[mask, j] > 0))

                            elif metric == "otiai":

                                sim = np.sum(train[mask, i] \* train[mask, j]) / (

                                    np.linalg.norm(train[mask, i])

                                    \* np.linalg.norm(train[mask, j])

                                )

                            else:

                                sim = 0

                        sims\_i.append(sim)

                        ratings\_i.append(train[u, j] - item\_means[j])

                    sims\_i = np.array(sims\_i)

                    ratings\_i = np.array(ratings\_i)

                    if np.sum(np.abs(sims\_i)) > 0:

                        preds[u, i] = item\_means[i] + np.dot(

                            sims\_i, ratings\_i

                        ) / np.sum(np.abs(sims\_i))

        mask\_eval = ~np.isnan(test\_true) & ~np.isnan(preds)

        if np.sum(mask\_eval) == 0:

            return np.nan, np.nan

        diff = preds[mask\_eval] - test\_true[mask\_eval]

        rmse = np.sqrt(np.mean(diff\*\*2))

        mae = np.mean(np.abs(diff))

        return rmse, mae

if \_\_name\_\_ == "\_\_main\_\_":

    cinema = MovieLensCinema(r"..\ml-latest-small")

    cinema.menu()

    # cinema.plot\_similarity\_metrics()