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 №2**

**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](#_Toc212571823)

[1 ПОСТАНОВКА ЗАДАЧИ 5](#_Toc212571824)

[2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 6](#_Toc212571825)

[2.1 Классификация марковских процессов 7](#_Toc212571826)

[2.2 Марковские случайные процессы с дискретными состояниями и дискретным временем 7](#_Toc212571827)

[2.3 Непрерывные цепи Маркова 9](#_Toc212571828)

[3 ДОКУМЕНТАЦИЯ К ДАННЫМ 11](#_Toc212571829)

[3.1 Описание предметной области 11](#_Toc212571830)

[3.2 Анализ данных 11](#_Toc212571831)

[3.3 Предобработка данных 15](#_Toc212571832)

[4 ПРАКТИЧЕСКАЯ ЧАСТЬ 16](#_Toc212571833)

[4.1 Функциональные возможности 16](#_Toc212571834)

[4.2 Дискретная Марковская цепь 20](#_Toc212571835)

[4.3 Непрерывная Марковская цепь 26](#_Toc212571836)

[ЗАКЛЮЧЕНИЕ 31](#_Toc212571837)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 32](#_Toc212571838)

[ПРИЛОЖЕНИЯ 33](#_Toc212571839)

# ВВЕДЕНИЕ

Современные стриминговые платформы и онлайн-кинотеатры ежедневно сталкиваются с проблемой эффективного управления вниманием пользователей в условиях практически неограниченного выбора контента. С расширением каталогов до десятков тысяч фильмов и сериалов традиционные методы ручного поиска становятся недостаточными для удовлетворения индивидуальных предпочтений пользователей. Проблема информационной перегрузки особенно остро стоит в сфере развлекательного контента, где пользователи тратят значительное время на поиск релевантных фильмов вместо непосредственного просмотра.

Актуальность разработки интеллектуальных рекомендательных систем подтверждается статистикой ведущих платформ: согласно исследованиям, более 80% просмотров на Netflix и 35% покупок на Amazon генерируются алгоритмами рекомендаций. Эффективные системы персонализации не только улучшают пользовательский опыт, но и напрямую влияют на ключевые бизнес-показатели, увеличивая удержание пользователей и монетизацию сервисов.

В настоящее время доминирующими подходами в рекомендательных системах являются методы коллаборативной фильтрации, основанные на поиске схожих пользователей (user-based) или объектов (item-based). Однако эти методы имеют фундаментальное ограничение — они не учитывают временную динамику предпочтений пользователей, рассматривая историю оценок как статичный набор данных. В реальности вкусы пользователей эволюционируют: переход от одних жанров к другим, сезонные предпочтения и изменение интересов со временем остаются за рамками традиционных подходов.

В данной работе предлагается использование марковских процессов для моделирования временной динамики пользовательских предпочтений. Марковские цепи позволяют учитывать не только что пользователь смотрел, но и в какой последовательности, а также временные интервалы между просмотрами. Такой подход особенно актуален для сервисов с большим объемом временных меток, где можно выявить паттерны переходов между жанрами и темами.

Для экспериментальной проверки эффективности подхода выбран датасет MovieLens, содержащий реальные оценки пользователей с временными метками. В работе реализованы и сравниваются два типа марковских моделей: дискретные цепи Маркова для моделирования переходов между жанрами и непрерывные цепи Маркова для учета временных интервалов между оценками.

Структура работы включает: анализ предметной области и данных, теоретическое обоснование марковских процессов, практическую реализацию рекомендательной системы с поддержкой различных стратегий рекомендаций, а также сравнительную оценку эффективности предложенного подхода. Особое внимание уделено визуализации марковских цепей и анализу матриц переходов, что обеспечивает прозрачность и интерпретируемость работы алгоритмов.

Практическая значимость работы заключается в разработке прототипа системы, способной адаптироваться к изменяющимся предпочтениям пользователей и учитывать временные паттерны их поведения, что может быть использовано для улучшения рекомендаций в реальных стриминговых платформах.

# 1 ПОСТАНОВКА ЗАДАЧИ

Цель работы: приобрести навыки реализации модельных коллаборативных методов рекомендательных систем, основанных на поиске скрытых факторов методом Марковских цепей.

Задачи: создать программную реализацию модельного метода рекомендательной системы (РС), основанной на дискретной и непрерывной Марковских цепях, включающую актуальную предметную область для применения РС (вроде маркетплейса, медиа ресурсов, соц. сетей, экономической сферы и т.д.) и набор начальных данных для неё (опрос покупателей, статистка за временной период, готовые вероятностные данные и т.д.), матрицу переходных вероятностей дискретной и матрицу плотностей вероятностей непрерывной Марковской цепи, строящуюся автоматически после ввода начальных данных (либо вводимую в качестве начальных данных), вектор начальных состояний, вывод состояния системы на шаге n для дискретной цепи или в случае с непрерывной цепью в момент времени t, свойств системы на основе матрицы переходных вероятностей (переходы, вероятности переходов и стационарности), графическую модель цепи (draw.io, yEd, Paint) с определёнными и подписанными элементами графа состояний системы (с обозначением состояний и их переходов в соответствии с тем, как они называются и располагаются в программной реализации).

# 2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Марковские случайные процессы названы по имени выдающегося русского математика А. А. Маркова (1856–1922), впервые начавшего изучение вероятностной связи случайных величин и создавшего теорию, которую можно назвать «динамикой вероятностей». В дальнейшем основы этой теории явились исходной базой общей теории случайных процессов, а также таких важных прикладных наук, как теория диффузионных процессов, теория надежности, теория массового обслуживания и, соответственно, в рекомендательных системах. Для математического описания многих операций, развивающихся в форме случайного процесса, может быть с успехом применен математический аппарат, разработанный в теории вероятностей для Марковских случайных процессов.

Функция 𝑋(𝑡) называется случайной, если ее значение при любом аргументе 𝑡 является случайной величиной. Случайная функция 𝑋(𝑡), аргументом которой является время, называется случайным процессом.

Марковские процессы являются частным видом случайных процессов. Особое место марковских процессов среди других классов случайных процессов обусловлено следующими обстоятельствами: для марковских процессов хорошо разработан математический аппарат, позволяющий решать многие практические задачи; с помощью марковских процессов можно описать (точно или приближенно) поведение достаточно сложных систем.

Случайный процесс, протекающий в какой-либо системе 𝑆, называется марковским (или процессом без последействия), если он обладает следующим свойством: для любою момента времени 𝑡0 вероятность любого состояния системы в будущем (при 𝑡 > 𝑡0) зависит только от ее состояния в настоящем (при 𝑡 = 𝑡0) и не зависит от того, когда и каким образом система 𝑆 пришла в это состояние. То есть в марковском случайном процессе будущее развитие процесса не зависит от его предыстории.

## 2.1 Классификация марковских процессов

Классификация марковских случайных процессов производится в зависимости от непрерывности или дискретности множества значений функции 𝑋(𝑡) и параметра 𝑡. Различают следующие основные виды марковских случайных процессов.

* с дискретными состояниями и дискретным временем (цепь Маркова);
* с непрерывными состояниями и дискретным временем (марковские последовательности);
* с дискретными состояниями и непрерывным временем (непрерывная цепь Маркова);
* с непрерывным состоянием и непрерывным временем.

## 2.2 Марковские случайные процессы с дискретными состояниями и дискретным временем

Если множество состояний, в которых может находиться процесс счётное, то есть все возможные состояния могут быть пронумерованы, то соответствующий процесс называется случайным процессом с дискретными состояниями или просто дискретным случайным процессом.

Марковские процессы с дискретными состояниями удобно иллюстрировать с помощью так называемого графа состояний, где кружками обозначены состояния 𝑆1, 𝑆2, …, Sn системы S, а стрелками – возможные переходы из состояния в состояние.

На графе отмечаются только непосредственные переходы, а не переходы через другие состояния. Возможные задержки в прежнем состоянии изображают «петлей», т. е. стрелкой, направленной из данного состояния в него же. Число состояний системы может быть, как конечным, так и бесконечным (но счетным).

Марковский случайный процесс с дискретными состояниями и дискретным временем называют марковской цепью. Для такого процесса моменты 𝑡1, 𝑡2, …, когда система 𝑆 может менять свое состояние, рассматривают как последовательные шаги процесса, а в качестве аргумента, от которого зависит процесс, выступает не время 𝑡, а номер шага 1, 2, . . ., 𝑘, ... Случайный процесс в этом случае характеризуется последовательностью состояний 𝑆(0), 𝑆(1), 𝑆(2), . . . , 𝑆(𝑘), . . ., где 𝑆(0) – начальное состояние системы (перед первым шагом); 𝑆(1) – состояние системы после первого шага; 𝑆(𝑘) – состояние системы после 𝑘-го шага.

Вероятностями состояний цепи Маркова называются вероятности того, что после 𝑘-го шага (и до (𝑘 + 1)-го) система 𝑆 будет находиться в состоянии . Очевидно, для любого 𝑘 выполняется условие, представленное в Формуле 2.1.

(2.1)

Начальным распределением вероятностей Марковской цепи называется распределение вероятностей состояний в начале процесса (Формула 2.2).

(2.2)

В частном случае, если начальное состояние системы 𝑆 в точности известно , то начальная вероятность , а все остальные равны нулю.

Вероятностью перехода (переходной вероятностью) на 𝑘-м шаге из состояния в состояние называется условная вероятность того, что система 𝑆 после 𝑘-го шага окажется в состоянии при условии, что непосредственно перед этим (после 𝑘 − 1 шага) она находилась в состоянии .

Поскольку система может пребывать в одном из 𝑛 состояний, то для каждого момента времени 𝑡 необходимо задать вероятностей перехода , которые удобно представить в виде матрицы (Формула 2.3).

(2.3)

где – вероятность перехода за один шаг из состояния в состояние ;

– вероятность задержки системы в состоянии .

Если переходные вероятности не зависят от номера шага (от времени), а зависят только от того, из какого состояния в какое осуществляется переход, то соответствующая цепь Маркова называется однородной.

## 2.3 Непрерывные цепи Маркова

Марковский случайный процесс с дискретными состояниями и непрерывным временем называется непрерывной цепью Маркова при условии, что переход системы из состояния в состояние происходит не в фиксированные, а в случайные моменты времени.

Пусть система характеризуется 𝑛 состояниями , а переход из состояния в состояние может осуществляться в любой момент времени. Обозначим через вероятность того, что в момент времени 𝑡 система 𝑆 будет находиться в состоянии . Требуется определить для любого 𝑡 вероятности состояний . Очевидно, что имеет место нормировочное условие (Формула 2.4).

(2.4)

Для процесса с непрерывным временем вместо переходных вероятностей рассматриваются плотности вероятностей перехода , представляющие coбой предел отношения вероятности перехода системы за время ∆𝑡 из состояния в состояние к длине промежутка ∆𝑡 (Формула 2.5).

(2.5)

где – вероятность того, что система, пребывавшая в момент 𝑡 в состоянии , за время ∆𝑡 перейдет из него в состояние .

Из определения плотностей вероятности перехода видно, что они в общем случае зависят от времени 𝑡, неотрицательны и в отличие от вероятностей могут быть больше 1.

Если при любых 𝑖 ≠ 𝑗 плотности вероятностей переходов не зависят от времени 𝑡, и тогда вместо будем писать просто , то Марковский процесс с непрерывным временем называется однородным. Если же хотя бы при одной паре значений 𝑖 ≠ 𝑗 плотность вероятности перехода изменяется с течением времени 𝑡, процесс называется неоднородным. Таким образом, если = const, то процесс называется однородным, если плотность вероятности зависит от времени = , то процесс – неоднородный.

Вероятности состояний (неизвестные вероятностные функции) являются решением системы дифференциальных уравнений (Формула 2.6).

(2.6)

Система представляет собой систему 𝑛 обыкновенных линейных однородных дифференциальных уравнений первого порядка с постоянными коэффициентами. Эта система называется системой дифференциальных уравнений Колмогорова. Величина называется потоком вероятности перехода из состояния в , причем интенсивность потоков может зависеть от времени или быть постоянной.

# 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.

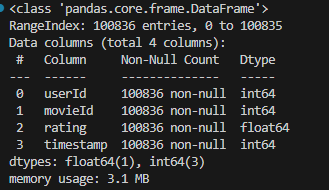


Рисунок 3.2.1 - Основная информация о данных в файле ratings.csv

Распределение выставленных оценок представлено на Рисунке 3.2.2.

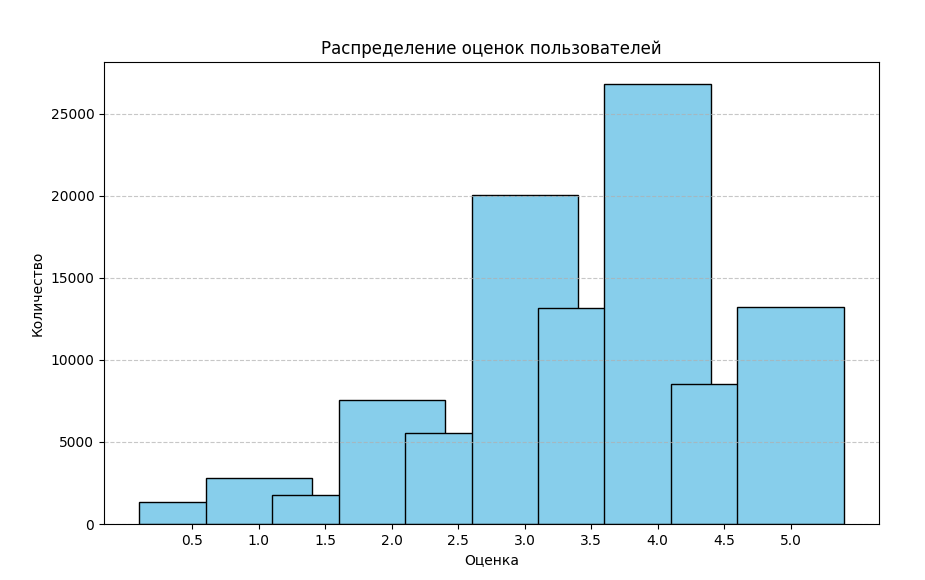


Рисунок 3.2.2 – Распределение оценок

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.

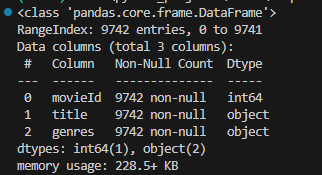


Рисунок 3.2.3 - Основная информация о данных в файле movies.csv

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

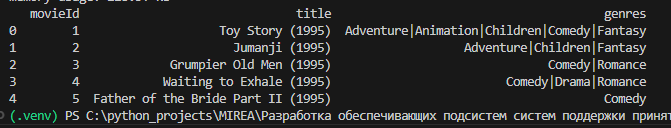


Рисунок 3.2.4 – Первые пять строк датафрейма movies

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.

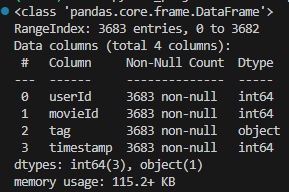


Рисунок 3.2.5 - Основная информация о данных в файле tags.csv

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.

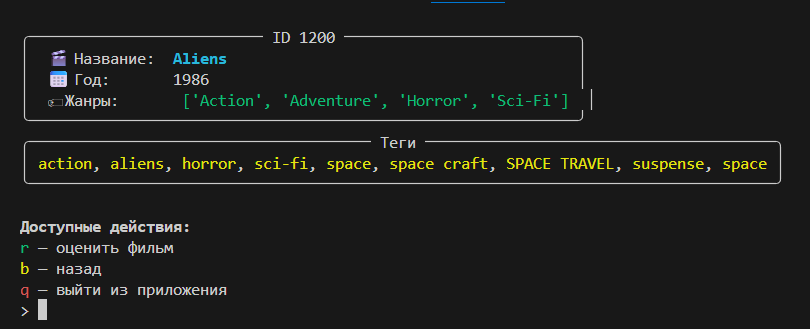


Рисунок 4.1.6 – Интерфейс страницы фильма

Пример выставления оценки показан на Рисунке 4.1.7.

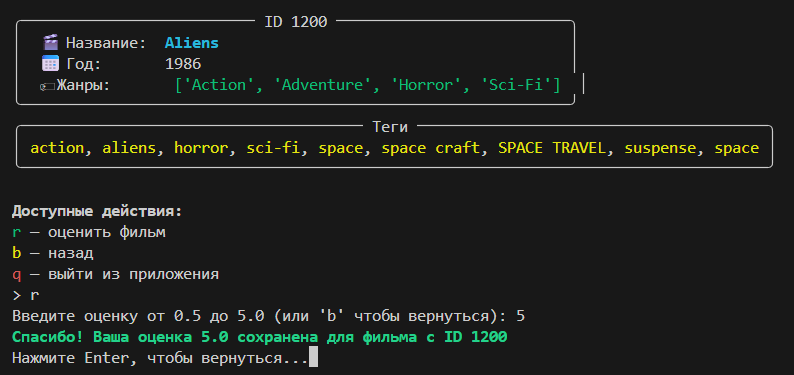


Рисунок 4.1.7 – Выставление оценки для фильма

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).

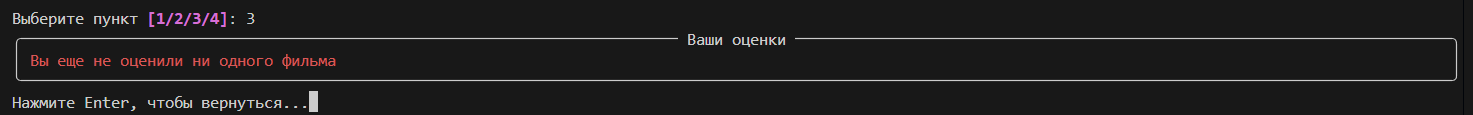


Рисунок 4.1.9 – Удаление оценки

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

## 4.2 Дискретная Марковская цепь

В рамках практической части реализована рекомендательная система, основанная на дискретной цепи Маркова. Данный подход позволяет моделировать переходы пользователя между различными жанрами фильмов на основе истории его оценок и предсказывать наиболее вероятные жанры для будущих просмотров.

Дискретная цепь Маркова реализована в отдельном классе MarkovChainRecommender, представленном в файле MarkovChain.py. Содержание файла представлено в Приложении Б.

В контексте рекомендательной системы состояния цепи соответствуют жанрам фильмов. Каждый фильм может принадлежать к нескольким жанрам, и при формировании последовательности оценок пользователя формируется цепочка переходов между жанрами.

После выставления пользователем хотя бы двух оценок система строит матрицу переходов, где элемент отражает вероятность перехода из жанра в жанр . Общий алгоритм построения матрицы переходов:

1. Извлекаются все фильмы, оцененные пользователем, в хронологическом порядке.
2. Для каждой последовательной пары фильмов (-й и -й) анализируются их жанры.
3. Если текущий фильм имеет несколько жанров, а следующий также имеет несколько жанров, вес перехода распределяется равномерно между всеми возможными переходами.

Переходы нормализуются так, чтобы сумма вероятностей по каждой строке была равна 1. Если для какого-то жанра нет исходящих переходов (например, пользователь оценил только один фильм этого жанра), для него устанавливается самопереход с вероятностью 1, что гарантирует корректность стохастической модели.

Рассмотрим процесс построения матрицы переходов на конкретном примере.

Предположим, что пользователь выставил оценки фильмам, представленным на Рисунке 4.2.1.

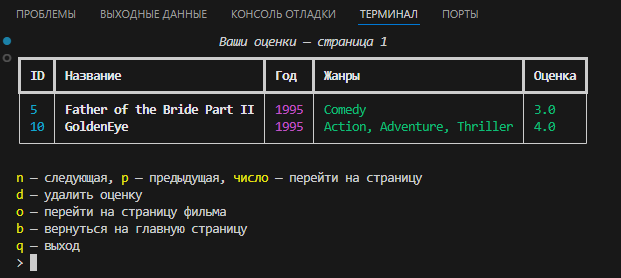


Рисунок 4.2.1 – Оценки пользователя

Сформировано множество всех жанров, встречающихся в оценённых фильмах (Формула 4.1).

(4.1)

Количество состояний после оценки двух фильмов: .

Матрица переходов инициализирована нулями в начальный момент времени (Формула 4.2).

(4.2)

Считаем, что индексация жанров такова, что они упорядочены по алфавиту ().

Каждый жанр из первого фильма порождает переход ко всем жанрам второго фильма с равной вероятностью, описанной Формулой 4.3.

(4.3)

Второй фильм содержит три жанра одновременно, значит вес одного перехода равен 1/3. Получены следующие вероятности переходов:

Все остальные элементы строки равны нулю, так как других переходов не было.

Далее выполняется нормализация каждой строки, чтобы сумма вероятностей по строке равнялась 1.

Для всех строк, где нет переходов (),  
устанавливается самопереход (Формула 4.4).

(4.4)

Итоговая матрица переходов задана Формулой 4.5, где строки и столбцы соответствуют жанрам в алфавитном порядке.

(4.5)

Программная реализация позволяет посмотреть на матрицу переходов после каждого шага k (выставление новой оценки, удаление старых оценок). Построенная матрица переходов представлена на Рисунке 4.2.2.

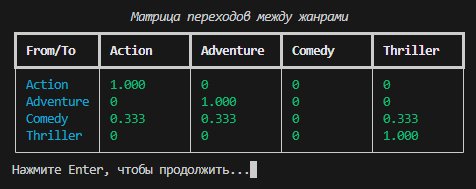


Рисунок 4.2.2 – Построенная матрица переходов

Программная реализация подтвердила корректность ручного расчета для матрицы переходов.

При генерации рекомендаций дополнительно выводится информация о последнем оцененном фильме, а также процентные соотношения для переходов в различные состояния (Рисунок 4.2.3).



Рисунок 4.2.3 – Вероятности жанров

На рисунке 4.2.3 показаны вероятности переходов из жанров последнего оцененного фильма. Процентные значения рассчитываются на основе умножения начального вектора на матрицу переходов в степени k (количество шагов), что позволяет прогнозировать предпочтения пользователя на несколько шагов вперед.

Выбор последнего оцененного фильма в качестве начального вектора обусловлен фундаментальным свойством марковских процессов - отсутствием последействия. Согласно определению цепи Маркова (Раздел 2.2), будущее состояние системы зависит только от текущего состояния и не зависит от предыстории. В контексте рекомендательной системы это означает, что следующий фильм, который захочет посмотреть пользователь, наиболее сильно зависит от его последних предпочтений, а не от всей истории просмотров. Таким образом, вектор начальных вероятностей сосредотачивается на жанрах последнего оцененного фильма, что соответствует марковскому свойству и обеспечивает релевантность рекомендаций.

Рекомендованные фильмы представлены на Рисунке 4.2.4.

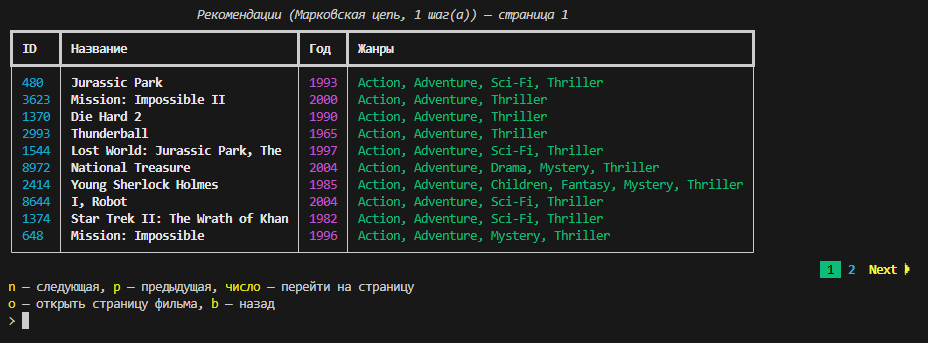


Рисунок 4.2.4 – Рекомендованные фильмы

На первых страницах рекомендаций выводятся фильмы, содержащие все три целевых жанра (Action, Adventure, Thriller), что соответствует максимальной суммарной вероятности переходов из последнего оцененного фильма, а на последних страницах – фильмы, содержащие лишь один из жанров, что соответствует одному состоянию, в которое пользователь может перейти.

На рисунке 4.2.5 представлена графическая модель дискретной цепи Маркова, автоматически сгенерированная системой.

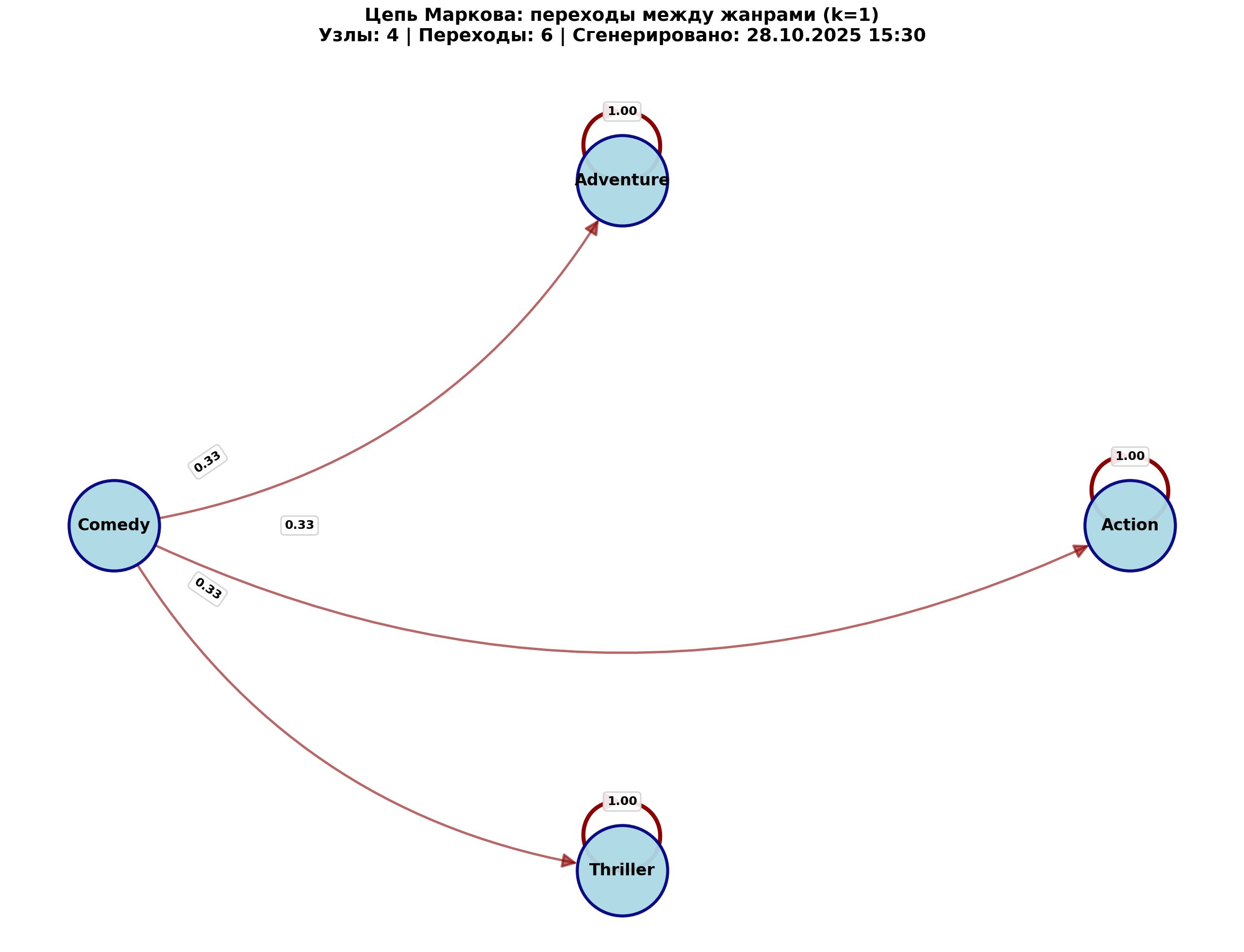


Рисунок 4.2.5 - Графическая модель состояний цепи

Граф визуализирует следующие элементы:

* узлы (круги): представляют состояния системы - жанры фильмов;
* ребра (стрелки): показывают возможные переходы между состояниями;
* веса переходов: числовые значения на стрелках отражают вероятности переходов между жанрами;
* петли: стрелки, направленные в тот же узел, обозначают вероятность остаться в текущем жанре.

В представленном графе четко видны рассчитанные ранее вероятности: из Comedy равновероятные переходы в Action, Adventure и Thriller (по 0.33), а для остальных жанров - самопереходы с вероятностью 1.0.

Практическая ценность данного подхода заключается в его способности улавливать динамику пользовательских предпочтений. В отличие от статических методов, цепь Маркова учитывает временную последовательность оценок, позволяя системе адаптироваться к изменяющимся вкусам пользователя. Например, если пользователь начинает переходить от комедий к триллерам, система своевременно скорректирует рекомендации в соответствии с этой тенденцией.

## 4.3 Непрерывная Марковская цепь

Дополнительно реализована рекомендательная система, основанная на непрерывной цепи Маркова. Данный подход позволяет учитывать временные интервалы между оценками пользователя и моделировать эволюцию его предпочтений в непрерывном времени, что обеспечивает более точные и адаптивные рекомендации.

Непрерывная цепь Маркова реализована в отдельном классе ContinuousMarkovRecommender, представленном в файле ContinuousMarkovChain.py (Приложение В).

Алгоритм построения матрицы интенсивностей в реализации включает следующие шаги:

1. Формирование хронологии оценок: оценки пользователя упорядочиваются по времени, при этом для оценок без временных меток генерируются реалистичные случайные интервалы.
2. Определение множества состояний: как и в дискретном случае, состояния соответствуют жанрам фильмов.
3. Расчет времени пребывания в состояниях: для каждого жанра вычисляется суммарное время, которое пользователь провел в этом состоянии.
4. Подсчет переходов между состояниями: анализируются последовательные пары оцененных фильмов и подсчитываются переходы между их жанрами.
5. Вычисление интенсивностей переходов: интенсивность перехода из состояния  в состояние  вычисляется по Формуле 4.6.

(4.6)

где – количество переходов из в состояние ;

– общее время пребывания в состоянии .

Рекомендации формируются на основе вычисленных вероятностей жанров с дополнительным учетом временного фактора. Алгоритм включает:

1. Вычисление вероятностей жанров для заданного времени .
2. Оценка релевантности каждого неоцененного фильма на основе совпадения жанров.
3. Учет временного фактора: для больших  увеличивается вес разнообразия жанров.
4. Ранжирование фильмов по итоговой оценке релевантности.

На Рисунке 4.3.1 представлены начальные оценки пользователя, на основе которых строится матрица интенсивностей.

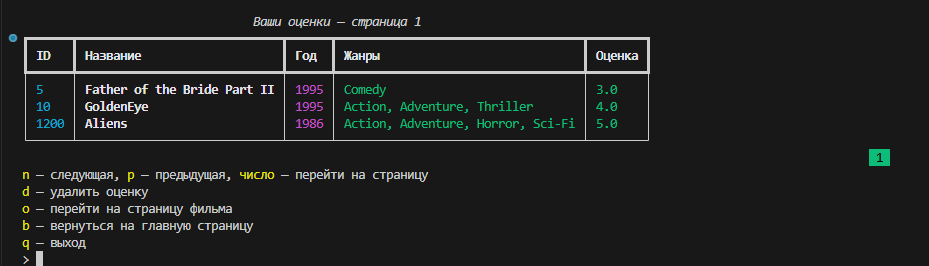


Рисунок 4.3.1 – Начальные оценки пользователя

Каждая оценка сопровождается временной меткой, что позволяет учитывать временные интервалы между просмотрами.

Реализована возможность задания времени для прогнозирования (в днях) и количества выдаваемых рекомендаций, как показано на Рисунке 4.3.2.

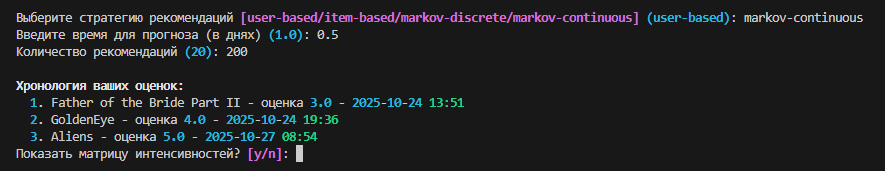


Рисунок 4.3.2 – Интерфейс настройки рекомендаций непрерывной цепью Маркова

Пользователь может задать параметр времени t (в днях) для прогнозирования и количество рекомендаций. Система также отображает хронологию оценок пользователя со сгенерированными временными метками.

Построенная матрица интенсивностей переходов представлена на Рисунке 4.3.3.



Рисунок 4.3.3 – Матрица интенсивности переходов

Значения на диагонали (отрицательные) представляют общую интенсивность выхода из состояния, а недиагональные элементы показывают интенсивности переходов между различными жанрами. Более высокие значения соответствуют более вероятным переходам.

Вероятности жанров для времени t=0.5 дней отображены на Рисунке 4.3.4.

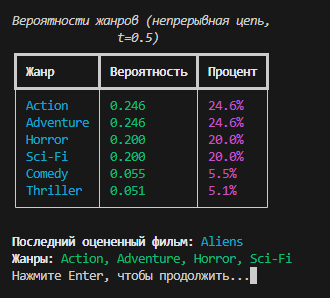


Рисунок 4.3.4 – Вероятности жанров

Таблица показывает распределение вероятностей по жанрам через 0.5 дней после последней оценки. Жанры с наибольшими вероятностями будут преобладать в рекомендациях.

Первая страница рекомендаций для параметра t=0.5 представлена на Рисунке 4.3.5.



Рисунок 4.3.5 – Первая страница рекомендаций с параметром t=0.5

Рекомендации сформированы на основе вероятностей жанров для короткого временного горизонта. Фильмы содержат жанры с наибольшими вероятностями, что обеспечивает релевантность рекомендаций текущим предпочтениям пользователя.

При увеличении временного горизонта до t=10 дней распределение вероятностей жанров изменяется, как показано на Рисунке 4.3.6.

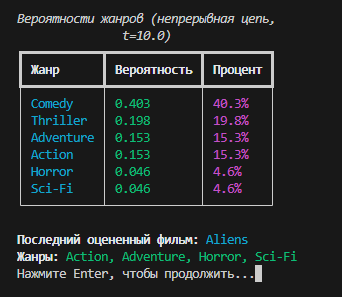


Рисунок 4.3.6 - Вероятности появления жанров для t=10 дней

Первая страница рекомендаций для параметра t=10 представлена на Рисунке 4.3.7.

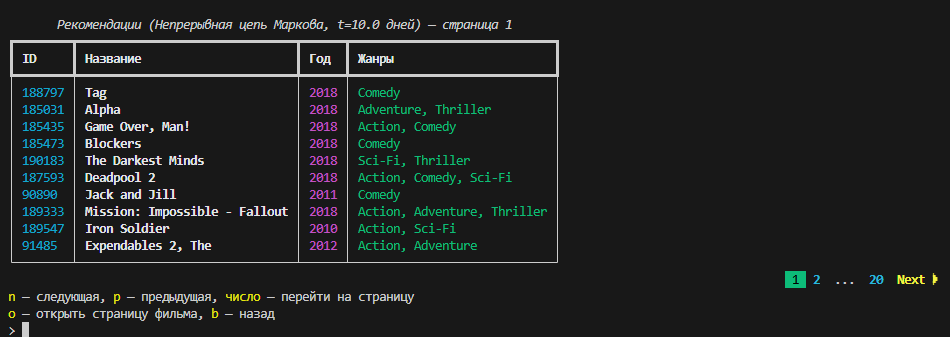


Рисунок 4.3.7 - Первая страница рекомендаций с параметром t=10

Рекомендации для большего временного горизонта включают более разнообразные жанры, что соответствует увеличенной неопределенности в предпочтениях пользователя в отдаленной перспективе.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной практической работы успешно достигнута поставленная цель: разработана и реализована рекомендательная система, основанная на методах марковских цепей, позволяющая моделировать динамику пользовательских предпочтений во времени. Все поставленные задачи выполнены в полном объеме.

Разработана функциональная консольная система на языке Python, интегрирующая интерактивный кинотеатр на основе датасета MovieLens и две ключевые стратегии рекомендаций на основе марковских процессов: дискретные цепи Маркова для моделирования переходов между жанрами и непрерывные цепи Маркова для учета временных интервалов между оценками. Система обеспечивает автоматическое построение матриц переходов и интенсивностей на основе пользовательских оценок, а также визуализацию графов состояний.

Проведенное исследование позволило выявить следующие ключевые особенности реализованных подходов.

Дискретные цепи Маркова показали высокую эффективность для моделирования непосредственных переходов между жанрами, обеспечивая релевантные рекомендации на основе последних оценок пользователя. Автоматически построенная матрица переходов точно отражает вероятности смены жанровых предпочтений, что подтверждается визуализацией графа состояний.

Непрерывные цепи Маркова продемонстрировали преимущество в задачах долгосрочного прогнозирования предпочтений, учитывая временные интервалы между оценками. Возможность задания временного горизонта прогнозирования позволяет системе адаптироваться к изменяющимся темпам потребления контента пользователем.

# 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: 24.10.2025).
4. Рекомендательные системы [Электронный ресурс]: URL: https://neerc.ifmo.ru/wiki/index.php?title=Рекомендательные\_системы (Дата обращения: 22.10.2025).
5. MovieLens [Электронный ресурс]: URL: https://grouplens.org/datasets/movielens/ (Дата обращения: 13.10.2025).
6. Краткое введение в цепи Маркова [Электронный ресурс]: URL: https://habr.com/ru/articles/455762/ (Дата обращения: 24.10.2025).
7. Марковская цепь [Электронный ресурс]: URL: <https://neerc.ifmo.ru/wiki/index.php?title=%D0%9C%D0%B0%D1%80%D0%BA%D0%BE%D0%B2%D1%81%D0%BA%D0%B0%D1%8F_%D1%86%D0%B5%D0%BF%D1%8C#.D0.A0.D0.B0.D1.81.D0.BF.D1.80.D0.B5.D0.B4.D0.B5.D0.BB.D0.B5.D0.BD.D0.B8.D0.B5_.D0.B2.D0.B5.D1.80.D0.BE.D1.8F.D1.82.D0.BD.D0.BE.D1.81.D1.82.D0.B5.D0.B9> (Дата обращения: 25.10.2025).

# ПРИЛОЖЕНИЯ

Приложение А — Реализация кинотеатра с встроенной рекомендательной системой, основанной на поиске скрытых факторов методом Марковских цепей.

Приложение Б — Код файла MarkovChain.py.

Приложение В — Код файла ContinuousMarkovChain.py.

### Приложение А

Реализация кинотеатра с встроенной рекомендательной системой, основанной на поиске скрытых факторов методом Марковских цепей

Листинг А – Реализация кинотеатра с встроенной рекомендательной системой, основанной на поиске скрытых факторов методом Марковских цепей

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

from datetime import datetime, timedelta

from MarkovChain import MarkovChainRecommender

from ContinuousMarkovChain import ContinuousMarkovRecommender

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, enable\_markov\_visualization=True):

        self.path = path

        self.links = None

        self.movies = None

        self.ratings = None

        self.tags = None

Продолжение Листинга А

        self.console = Console()

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

        self.load\_data()

        self.per\_page = per\_page

        self.markov\_recommender = MarkovChainRecommender(

            self, auto\_visualize=enable\_markov\_visualization

        )

        self.continuous\_markov\_recommender = ContinuousMarkovRecommender(

            self, auto\_visualize=enable\_markov\_visualization

    )

    def load\_data(self):

        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"])

Продолжение Листинга А

            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",

                },

                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(

Продолжение Листинга А

                "[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:

                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":

Продолжение Листинга А

                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]")

            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"

Продолжение Листинга А

            )

            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:

            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.markov\_recommender.reset\_matrix()

                    self.continuous\_markov\_recommender.reset\_matrix()

                    self.markov\_recommender.build\_transition\_matrix()

                    self.continuous\_markov\_recommender.build\_intensity\_matrix()

                    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:

Продолжение Листинга А

                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")

            )

            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(

Продолжение Листинга А

                "[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:

                    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()

Продолжение Листинга А

        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="=",

        )

        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"

                )

            )

Продолжение Листинга А

            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")

            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)"

                )

Продолжение Листинга А

                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)

                        self.markov\_recommender.reset\_matrix()

                        self.markov\_recommender.build\_transition\_matrix()

                    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)")

                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

Продолжение Листинга А

        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 ""

            )

            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"

Продолжение Листинга А

                "[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":

                break

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

    def show\_recommendations(self):

        """Меню выбора стратегии рекомендаций"""

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

        if self.rates.empty:

            self.console.print(

                Panel(

                    "[red]У вас нет оценок для генерации рекомендаций.[/red]",

                    title="Рекомендации",

                )

            )

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

            return

        strategy = Prompt.ask(

            "Выберите стратегию рекомендаций",

            choices=["user-based", "item-based", "markov-discrete", "markov-continuous"],

            default="user-based",

        )

        if strategy == "user-based":

            self.recommend\_user\_based()

        elif strategy == "item-based":

            self.recommend\_item\_based()

        elif strategy == "markov-discrete":

            self.recommend\_markov\_chain()

        else:

            self.recommend\_continuous\_markov\_chain()

    def recommend\_user\_based(self):

        """User-based рекомендации"""

        R = self.get\_rating\_matrix()

        user\_ids = R.index.tolist()

        movie\_ids = R.columns.tolist()

        current\_user\_id = -1

        current\_user\_ratings = pd.Series(

Продолжение Листинга А

            [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"]

        R = pd.concat(

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

        )

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

        metric = Prompt.ask(

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

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

            default="pearson",

        )

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

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

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

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

        sims = np.zeros(n\_users)

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

        for u in range(n\_users):

            if u == current\_user\_idx:

                continue

            other = R\_values[u, :]

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

            if np.sum(mask) == 0:

                continue

            if metric == "pearson":

                sims[u] = np.corrcoef(current\_ratings[mask], other[mask])[0, 1]

            elif metric == "lp":

                sims[u] = -np.linalg.norm(current\_ratings[mask] - other[mask])

            elif metric == "jaccard":

                sims[u] = np.sum(

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

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

            elif metric == "otiai":

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

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

                )

        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

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

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

        self.paginated\_view(

            self.movies.loc[recs\_df["movieId"]], title=f"User-based ({metric})"

Продолжение Листинга А

        )

    def recommend\_item\_based(self):

        """Item-based рекомендации"""

        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"]

        R = pd.concat(

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

        )

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

        metric = Prompt.ask(

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

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

            default="pearson",

        )

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

        n\_items = R\_values.shape[1]

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

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

        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]

                    elif metric == "lp":

                        sim = -np.linalg.norm(R\_values[mask, i] - R\_values[mask, j])

                    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.dot(R\_values[mask, i], R\_values[mask, j]) / (

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

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

                        )

                sims\_i.append(sim)

Продолжение Листинга А

                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": R.columns, "pred\_rating": preds})

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

        self.paginated\_view(

            self.movies.loc[recs\_df["movieId"]], title=f"Item-based ({metric})"

        )

    def recommend\_markov\_chain(self):

        """Улучшенные рекомендации с помощью дискретной цепи Маркова"""

        if self.rates.empty:

            self.console.print(

                Panel("[red]Вы не оценили ни одного фильма![/red]", title="Ошибка")

            )

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

            return

        if len(self.rates) < 2:

            self.console.print(

                Panel(

                    "[yellow]Оцените хотя бы 2 фильма для построения цепи Маркова[/yellow]",

                    title="Недостаточно данных",

                )

            )

            unrated\_movies = self.movies[~self.movies.index.isin(self.rates.index)]

            if not unrated\_movies.empty:

                if Confirm.ask("Показать случайные рекомендации вместо цепи Маркова?"):

                    recommendations = unrated\_movies.sample(

                        min(20, len(unrated\_movies))

                    )

                    self.paginated\_view(recommendations, title="Случайные рекомендации")

            else:

                self.console.print("[red]Нет фильмов для рекомендаций[/red]")

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

            return

        markov\_rec = self.markov\_recommender

        steps = Prompt.ask(

            "Количество шагов цепи Маркова", choices=["1", "2", "3"], default="1"

        )

        steps = int(steps)

        top\_k = Prompt.ask("Количество рекомендаций", default="20")

        top\_k = int(top\_k)

        recommendations = markov\_rec.recommend\_movies(steps=steps, top\_k=top\_k)

Продолжение Листинга А

        if recommendations.empty:

            self.console.print(

                Panel(

                    "[yellow]Не найдено подходящих рекомендаций. Попробуйте оценить больше фильмов разных жанров.[/yellow]",

                    title="Рекомендации",

                )

            )

            unrated\_movies = self.movies[~self.movies.index.isin(self.rates.index)]

            if not unrated\_movies.empty:

                if Confirm.ask("Показать случайные рекомендации?"):

                    random\_recs = unrated\_movies.sample(min(20, len(unrated\_movies)))

                    self.paginated\_view(random\_recs, title="Случайные рекомендации")

            else:

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

            return

        if Confirm.ask("Показать матрицу переходов?"):

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

            markov\_rec.show\_transition\_matrix()

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

        if Confirm.ask("Показать вероятности жанров?"):

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

            self.show\_markov\_genre\_probabilities(steps)

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

        self.console.print(

            Panel(

                f"🎲 Рекомендации на основе цепи Маркова ({steps} шаг(а))",

                title="Цепь Маркова",

            )

        )

        self.paginated\_view(

            recommendations, title=f"Рекомендации (Марковская цепь, {steps} шаг(а))"

        )

    def show\_markov\_genre\_probabilities(self, steps=1):

        """Показать вероятности жанров для цепи Маркова"""

        genre\_probs = self.markov\_recommender.get\_genre\_probabilities(steps)

        if not genre\_probs:

            self.console.print(

                "[red]Не удалось вычислить вероятности жанров. Оцените больше фильмов для построения цепи Маркова.[/red]"

            )

            return

        sorted\_genres = sorted(genre\_probs.items(), key=lambda x: x[1], reverse=True)

        prob\_table = Table(title=f"Вероятности жанров (через {steps} шаг(а))")

        prob\_table.add\_column("Жанр", style="cyan")

        prob\_table.add\_column("Вероятность", style="green")

Продолжение Листинга А

        prob\_table.add\_column("Процент", style="magenta")

        for genre, prob in sorted\_genres:

            if prob > 0.001:

                percentage = prob \* 100

                prob\_table.add\_row(genre, f"{prob:.3f}", f"{percentage:.1f}%")

        self.console.print(prob\_table)

        rated\_movies = self.movies.loc[self.rates.index]

        if not rated\_movies.empty:

            last\_movie = rated\_movies.iloc[-1]

            self.console.print(

                f"\n[bold]Последний оцененный фильм:[/bold] [cyan]{last\_movie['title']}[/cyan]"

            )

            self.console.print(

                f"[bold]Жанры:[/bold] [green]{', '.join(last\_movie['genres'])}[/green]"

            )

    def get\_ratings\_with\_timestamps(self):

        """Получение оценок с временными метками в хронологическом порядке"""

        if self.rates.empty:

            return []

        ratings\_list = []

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

            if 'datetime' in row:

                timestamp = row['datetime']

            else:

                timestamp = datetime.now() - timedelta(days=len(ratings\_list))

            ratings\_list.append((movie\_id, row['rating'], timestamp))

        ratings\_list.sort(key=lambda x: x[2])

        return ratings\_list

    def recommend\_continuous\_markov\_chain(self):

        """Рекомендации на основе непрерывной цепи Маркова с временными метками"""

        if self.rates.empty:

            self.console.print(

                Panel("[red]Вы не оценили ни одного фильма![/red]", title="Ошибка")

            )

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

            return

        if len(self.rates) < 2:

            self.console.print(

                Panel(

                    "[yellow]Оцените хотя бы 2 фильма для построения непрерывной цепи Маркова[/yellow]",

                    title="Недостаточно данных",

                )

            )

            return

        continuous\_markov = self.continuous\_markov\_recommender

Продолжение Листинга А

        time\_t = Prompt.ask(

            "Введите время для прогноза (в днях)", default="1.0"

        )

        try:

            time\_t = float(time\_t)

        except ValueError:

            time\_t = 1.0

        top\_k = Prompt.ask("Количество рекомендаций", default="20")

        top\_k = int(top\_k)

        ratings\_with\_time = self.continuous\_markov\_recommender.get\_ratings\_chronology()

        if ratings\_with\_time:

            self.console.print("\n[bold]Хронология ваших оценок:[/bold]")

            for i, (movie\_id, rating, timestamp) in enumerate(ratings\_with\_time):

                movie\_title = self.movies.loc[movie\_id, "title"]

                time\_str = timestamp.strftime("%Y-%m-%d %H:%M") if isinstance(timestamp, datetime) else "недавно"

                self.console.print(f"  {i+1}. {movie\_title} - оценка {rating} - {time\_str}")

        recommendations = continuous\_markov.recommend\_movies\_continuous(

            time\_t=time\_t, top\_k=top\_k

        )

        if recommendations.empty:

            self.console.print(

                Panel(

                    "[yellow]Не найдено подходящих рекомендаций[/yellow]",

                    title="Рекомендации",

                )

            )

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

            return

        if Confirm.ask("Показать матрицу интенсивностей?"):

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

            continuous\_markov.show\_intensity\_matrix()

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

        if Confirm.ask("Показать вероятности жанров для непрерывной цепи?"):

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

            self.show\_continuous\_markov\_genre\_probabilities(time\_t)

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

        self.console.print(

            Panel(

                f"⏱️ Рекомендации на основе непрерывной цепи Маркова (t={time\_t} дней)",

                title="Непрерывная цепь Маркова",

            )

        )

        self.paginated\_view(

            recommendations,

            title=f"Рекомендации (Непрерывная цепь Маркова, t={time\_t} дней)"

        )

Продолжение Листинга А

    def show\_continuous\_markov\_genre\_probabilities(self, time\_t=1.0):

        """Показать вероятности жанров для непрерывной цепи Маркова"""

        genre\_probs = self.continuous\_markov\_recommender.get\_genre\_probabilities\_continuous(time\_t)

        if not genre\_probs:

            self.console.print(

                "[red]Не удалось вычислить вероятности жанров для непрерывной цепи.[/red]"

            )

            return

        sorted\_genres = sorted(genre\_probs.items(), key=lambda x: x[1], reverse=True)

        prob\_table = Table(title=f"Вероятности жанров (непрерывная цепь, t={time\_t})")

        prob\_table.add\_column("Жанр", style="cyan")

        prob\_table.add\_column("Вероятность", style="green")

        prob\_table.add\_column("Процент", style="magenta")

        for genre, prob in sorted\_genres:

            if prob > 0.001:

                percentage = prob \* 100

                prob\_table.add\_row(genre, f"{prob:.3f}", f"{percentage:.1f}%")

        self.console.print(prob\_table)

        rated\_movies = self.movies.loc[self.rates.index]

        if not rated\_movies.empty:

            last\_movie = rated\_movies.iloc[-1]

            self.console.print(

                f"\n[bold]Последний оцененный фильм:[/bold] [cyan]{last\_movie['title']}[/cyan]"

            )

            self.console.print(

                f"[bold]Жанры:[/bold] [green]{', '.join(last\_movie['genres'])}[/green]"

            )

    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", enable\_markov\_visualization=True)

    cinema.menu()

    # cinema.plot\_similarity\_metrics()

### Приложение Б

Код файла MarkovChain.py

Листинг Б – Код файла MarkovChain.py

import numpy as np

import os

import shutil

import networkx as nx

import matplotlib.pyplot as plt

from datetime import datetime

from rich.table import Table

class MarkovChainRecommender:

    """Внутренний класс для рекомендаций на основе цепи Маркова"""

    def \_\_init\_\_(self, cinema, auto\_visualize=True):

        self.cinema = cinema

        self.transition\_matrix = None

        self.genre\_states = []

        self.auto\_visualize = auto\_visualize

        self.\_clean\_markov\_directory()

    def \_clean\_markov\_directory(self):

        """Очистка папки markov при запуске программы"""

        try:

            if os.path.exists("markov"):

                shutil.rmtree("markov")

            os.makedirs("markov", exist\_ok=True)

        except Exception as e:

            pass

    def reset\_matrix(self):

        """Сброс матрицы переходов (вызывается при изменении оценок)"""

        self.transition\_matrix = None

        self.genre\_states = []

    def build\_transition\_matrix(self):

        """Построение матрицы переходов между жанрами с корректной нормализацией (вариант 1 — честная модель)"""

        if self.cinema.rates.empty:

            return False

        rated\_movies = self.cinema.movies.loc[self.cinema.rates.index]

        all\_genres = set()

        for genres in rated\_movies["genres"]:

            all\_genres.update(genres)

        self.genre\_states = sorted(all\_genres)

        if not self.genre\_states:

            return False

        n\_genres = len(self.genre\_states)

        self.transition\_matrix = np.zeros((n\_genres, n\_genres))

        state\_to\_idx = {genre: idx for idx, genre in enumerate(self.genre\_states)}

        rated\_indices = list(rated\_movies.index)

Продолжение Листинга Б

        if len(rated\_indices) < 2:

            return False

        for i in range(len(rated\_indices) - 1):

            current\_movie\_id = rated\_indices[i]

            next\_movie\_id = rated\_indices[i + 1]

            current\_genres = rated\_movies.loc[current\_movie\_id, "genres"]

            next\_genres = rated\_movies.loc[next\_movie\_id, "genres"]

            if not current\_genres or not next\_genres:

                continue

            weight = 1.0 / len(next\_genres)

            for curr\_genre in current\_genres:

                if curr\_genre not in state\_to\_idx:

                    continue

                curr\_idx = state\_to\_idx[curr\_genre]

                for next\_genre in next\_genres:

                    if next\_genre not in state\_to\_idx:

                        continue

                    next\_idx = state\_to\_idx[next\_genre]

                    self.transition\_matrix[curr\_idx, next\_idx] += weight

        row\_sums = self.transition\_matrix.sum(axis=1)

        for i in range(n\_genres):

            if row\_sums[i] > 0:

                self.transition\_matrix[i] /= row\_sums[i]

            else:

                self.transition\_matrix[i, i] = 1.0

        if self.auto\_visualize:

            self.\_auto\_visualize\_markov\_chain()

        return True

    def get\_genre\_probabilities(self, steps=1):

        """Получение вероятностей жанров через steps шагов"""

        if self.transition\_matrix is None:

            success = self.build\_transition\_matrix()

            if not success:

                return None

        if self.cinema.rates.empty:

            return None

        rated\_movies = self.cinema.movies.loc[self.cinema.rates.index]

        last\_movie\_id = rated\_movies.index[-1]

        last\_genres = rated\_movies.loc[last\_movie\_id, "genres"]

        initial\_vector = np.zeros(len(self.genre\_states))

        genre\_count = len(last\_genres)

        if genre\_count == 0:

            return None

        for genre in last\_genres:

            if genre in self.genre\_states:

Продолжение Листинга Б

                idx = self.genre\_states.index(genre)

                initial\_vector[idx] = 1.0 / genre\_count

        current\_probs = initial\_vector

        for \_ in range(steps):

            current\_probs = current\_probs @ self.transition\_matrix

        return dict(zip(self.genre\_states, current\_probs))

    def recommend\_movies(self, steps=1, top\_k=20, min\_probability=0.01):

        """Рекомендация фильмов на основе цепи Маркова"""

        genre\_probs = self.get\_genre\_probabilities(steps)

        if not genre\_probs:

            unrated\_movies = self.cinema.movies[

                ~self.cinema.movies.index.isin(self.cinema.rates.index)

            ]

            return unrated\_movies.sample(min(20, len(unrated\_movies)))

        significant\_genres = [

            genre for genre, prob in genre\_probs.items() if prob >= min\_probability

        ]

        if not significant\_genres:

            significant\_genres = [max(genre\_probs, key=genre\_probs.get)]

        candidate\_movies = self.cinema.movies[

            ~self.cinema.movies.index.isin(self.cinema.rates.index)

        ].copy()

        if candidate\_movies.empty:

            return candidate\_movies

        def calculate\_score(genres):

            return sum(genre\_probs.get(genre, 0) for genre in genres)

        candidate\_movies["markov\_score"] = candidate\_movies["genres"].apply(

            calculate\_score

        )

        recommendations = (

            candidate\_movies[candidate\_movies["markov\_score"] > 0]

            .sort\_values("markov\_score", ascending=False)

            .head(top\_k)

        )

        return recommendations.drop(columns=["markov\_score"], errors="ignore")

    def show\_transition\_matrix(self):

        """Отображение матрицы переходов"""

        if self.transition\_matrix is None:

            success = self.build\_transition\_matrix()

            if not success:

                self.cinema.console.print(

                    "[red]Не удалось построить матрицу переходов[/red]"

                )

                return

        table = Table(title="Матрица переходов между жанрами")

        table.add\_column("From/To", style="cyan")

Продолжение Листинга Б

        for genre in self.genre\_states:

            table.add\_column(genre, style="green", width=10)

        for i, from\_genre in enumerate(self.genre\_states):

            row = [from\_genre]

            for j, to\_genre in enumerate(self.genre\_states):

                prob = self.transition\_matrix[i, j]

                if prob > 0:

                    row.append(f"{prob:.3f}")

                else:

                    row.append("0")

            table.add\_row(\*row)

        self.cinema.console.print(table)

    def \_auto\_visualize\_markov\_chain(self, steps=1):

        """Автоматическое построение и сохранение графа цепи Маркова с финальными настройками"""

        os.makedirs("markov", exist\_ok=True)

        if self.transition\_matrix is None or len(self.genre\_states) == 0:

            return

        G = nx.DiGraph()

        for genre in self.genre\_states:

            G.add\_node(genre)

        edge\_labels = {}

        for i, from\_genre in enumerate(self.genre\_states):

            for j, to\_genre in enumerate(self.genre\_states):

                prob = self.transition\_matrix[i, j]

                if prob >= 0.001:

                    G.add\_edge(from\_genre, to\_genre, weight=prob)

                    edge\_labels[(from\_genre, to\_genre)] = f"{prob:.2f}"

        plt.figure(figsize=(18, 14), dpi=120)

        if len(self.genre\_states) <= 6:

            pos = nx.circular\_layout(G, scale=3.0)

        elif len(self.genre\_states) <= 10:

            pos = nx.circular\_layout(G, scale=3.5)

        else:

            pos = nx.spring\_layout(G, k=4, iterations=150, scale=3)

        node\_size = 8000

        nx.draw\_networkx\_nodes(

            G,

            pos,

            node\_size=node\_size,

            node\_color="lightblue",

            alpha=0.95,

            edgecolors="navy",

            linewidths=3,

            node\_shape="o",

        )

        nx.draw\_networkx\_labels(

            G,

            pos,

            font\_size=16,

Продолжение Листинга Б

            font\_weight="bold",

            font\_family="DejaVu Sans",

            verticalalignment="center",

            horizontalalignment="center",

        )

        if G.edges():

            edge\_weights = [G[u][v]["weight"] for u, v in G.edges()]

            max\_weight = max(edge\_weights) if edge\_weights else 1

            edge\_widths = [1.5 + (w / max\_weight) \* 2.5 for w in edge\_weights]

            edge\_alphas = [0.4 + (w / max\_weight) \* 0.6 for w in edge\_weights]

            edges = nx.draw\_networkx\_edges(

                G,

                pos,

                edge\_color="darkred",

                width=edge\_widths,

                alpha=edge\_alphas,

                arrows=True,

                arrowsize=30,

                arrowstyle="-|>",

                connectionstyle="arc3,rad=0.25",

                min\_source\_margin=20,

                min\_target\_margin=25,

                node\_size=node\_size,

                ax=plt.gca(),

            )

            nx.draw\_networkx\_edge\_labels(

                G,

                pos,

                edge\_labels=edge\_labels,

                font\_size=12,

                font\_weight="bold",

                font\_family="DejaVu Sans",

                label\_pos=0.18,

                verticalalignment="center",

                horizontalalignment="center",

                bbox=dict(

                    boxstyle="round,pad=0.3",

                    facecolor="white",

                    alpha=0.9,

                    edgecolor="lightgray",

                    linewidth=1.5,

                ),

            )

        title\_info = f"Цепь Маркова: переходы между жанрами (k={steps})\n"

        title\_info += f"Узлы: {len(G.nodes())} | Переходы: {len(G.edges())} | "

        title\_info += f"Сгенерировано: {datetime.now().strftime('%d.%m.%Y %H:%M')}"

        plt.title(

            title\_info,

            fontsize=18,

            fontweight="bold",

            pad=30,

            loc="center",

            fontfamily="DejaVu Sans",

        )

Окончание Листинга Б

        plt.axis("off")

        plt.tight\_layout(pad=4.0)

        timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")

        filename = f"markov/markov\_chain\_k{steps}\_{timestamp}.png"

        plt.savefig(

            filename, dpi=150, bbox\_inches="tight", facecolor="white", edgecolor="none"

        )

        plt.close()

### Приложение В

Код файла ContinuousMarkovChain.py

Листинг В – Код файла ContinuousMarkovChain.py

import numpy as np

import pandas as pd

from scipy.linalg import expm

from rich.table import Table

import os

import matplotlib.pyplot as plt

import networkx as nx

from datetime import datetime, timedelta

class ContinuousMarkovRecommender:

    """Исправленная реализация непрерывной цепи Маркова с реальными временными метками"""

    def \_\_init\_\_(self, cinema, auto\_visualize=True):

        self.cinema = cinema

        self.intensity\_matrix = None

        self.genre\_states = []

        self.auto\_visualize = auto\_visualize

        self.time\_unit = 1.0

    def reset\_matrix(self):

        """Сброс матрицы интенсивностей"""

        self.intensity\_matrix = None

        self.genre\_states = []

    def build\_intensity\_matrix(self):

        """Построение матрицы интенсивностей с использованием реальных данных"""

        if self.cinema.rates.empty or len(self.cinema.rates) < 2:

            return False

        ratings\_chronology = self.get\_ratings\_chronology()

        if len(ratings\_chronology) < 2:

            return False

        all\_genres = set()

        for movie\_id, rating, time\_info in ratings\_chronology:

            movie\_genres = self.cinema.movies.loc[movie\_id, "genres"]

            all\_genres.update(movie\_genres)

        self.genre\_states = sorted(all\_genres)

        if not self.genre\_states:

            return False

        n\_genres = len(self.genre\_states)

        state\_to\_idx = {genre: idx for idx, genre in enumerate(self.genre\_states)}

        time\_in\_state = {genre: 0.0 for genre in self.genre\_states}

        transition\_counts = np.zeros((n\_genres, n\_genres))

        for i in range(len(ratings\_chronology) - 1):

            current\_movie\_id, current\_rating, current\_time = ratings\_chronology[i]

            next\_movie\_id, next\_rating, next\_time = ratings\_chronology[i + 1]

            current\_genres = self.cinema.movies.loc[current\_movie\_id, "genres"]

Продолжение Листинга В

            next\_genres = self.cinema.movies.loc[next\_movie\_id, "genres"]

            if not current\_genres or not next\_genres:

                continue

            time\_interval = self.calculate\_time\_interval(current\_time, next\_time)

            for genre in current\_genres:

                if genre in state\_to\_idx:

                    time\_in\_state[genre] += time\_interval / len(current\_genres)

            for curr\_genre in current\_genres:

                if curr\_genre not in state\_to\_idx:

                    continue

                curr\_idx = state\_to\_idx[curr\_genre]

                for next\_genre in next\_genres:

                    if next\_genre not in state\_to\_idx:

                        continue

                    next\_idx = state\_to\_idx[next\_genre]

                    weight = 1.0 / (len(current\_genres) \* len(next\_genres))

                    transition\_counts[curr\_idx, next\_idx] += weight

        self.intensity\_matrix = np.zeros((n\_genres, n\_genres))

        for i, genre\_i in enumerate(self.genre\_states):

            total\_time = time\_in\_state[genre\_i]

            if total\_time > 0:

                for j, genre\_j in enumerate(self.genre\_states):

                    if i != j and transition\_counts[i, j] > 0:

                        self.intensity\_matrix[i, j] = (

                            transition\_counts[i, j] / total\_time

                        )

            self.intensity\_matrix[i, i] = -np.sum(self.intensity\_matrix[i])

        symmetric\_matrix = (self.intensity\_matrix + self.intensity\_matrix.T) / 2

        for i in range(n\_genres):

            for j in range(n\_genres):

                if i != j:

                    self.intensity\_matrix[i, j] = max(symmetric\_matrix[i, j], 0)

            self.intensity\_matrix[i, i] = -np.sum(self.intensity\_matrix[i])

        min\_intensity = 0.01

        for i in range(n\_genres):

            if abs(self.intensity\_matrix[i, i]) < min\_intensity:

                for j in range(n\_genres):

                    if i != j:

                        self.intensity\_matrix[i, j] += min\_intensity / (n\_genres - 1)

                self.intensity\_matrix[i, i] = -np.sum(self.intensity\_matrix[i])

        if self.auto\_visualize:

            self.\_auto\_visualize\_markov\_chain()

        return True

    def get\_ratings\_chronology(self, random\_intervals=True):

Продолжение Листинга В

        """Создание хронологии оценок с реалистичными случайными интервалами"""

        if self.cinema.rates.empty:

            return []

        ratings\_list = []

        current\_time = datetime.now()

        prev\_time = current\_time - timedelta(days=len(self.cinema.rates) \* 2)

        for i, (movie\_id, row) in enumerate(self.cinema.rates.iterrows()):

            if random\_intervals:

                interval\_days = np.random.exponential(scale=2.0)

            else:

                interval\_days = 1.0

            timestamp = prev\_time + timedelta(days=interval\_days)

            ratings\_list.append((movie\_id, row["rating"], timestamp))

            prev\_time = timestamp

        return sorted(ratings\_list, key=lambda x: x[2])

    def calculate\_time\_interval(self, time1, time2):

        """Вычисление временного интервала между двумя оценками"""

        time\_diff = abs((time2 - time1).total\_seconds()) / (24 \* 3600)

        return max(time\_diff, 0.1)

    def get\_genre\_probabilities\_continuous(self, time\_t):

        """Получение вероятностей жанров в момент времени t с нормализацией"""

        if self.intensity\_matrix is None:

            success = self.build\_intensity\_matrix()

            if not success:

                return None

        if self.cinema.rates.empty:

            return None

        ratings\_chronology = self.get\_ratings\_chronology()

        if not ratings\_chronology:

            return None

        last\_movie\_id, last\_rating, last\_time = ratings\_chronology[-1]

        last\_genres = self.cinema.movies.loc[last\_movie\_id, "genres"]

        initial\_vector = np.zeros(len(self.genre\_states))

        genre\_count = len(last\_genres)

        if genre\_count == 0:

            return None

        for genre in last\_genres:

            if genre in self.genre\_states:

                idx = self.genre\_states.index(genre)

                initial\_vector[idx] = 1.0 / genre\_count

        try:

            transition\_matrix = expm(self.intensity\_matrix \* time\_t)

            current\_probs = initial\_vector @ transition\_matrix

            prob\_sum = np.sum(current\_probs)

            if prob\_sum > 0:

                current\_probs = current\_probs / prob\_sum

Продолжение Листинга В

            else:

                current\_probs = np.ones\_like(current\_probs) / len(current\_probs)

        except Exception as e:

            print(f"Ошибка вычисления матричной экспоненты: {e}")

            return None

        return dict(zip(self.genre\_states, current\_probs))

    def recommend\_movies\_continuous(self, time\_t=1.0, top\_k=20):

        """Улучшенные рекомендации, чувствительные к времени"""

        genre\_probs = self.get\_genre\_probabilities\_continuous(time\_t)

        if not genre\_probs:

            unrated\_movies = self.cinema.movies[

                ~self.cinema.movies.index.isin(self.cinema.rates.index)

            ]

            return unrated\_movies.sample(min(top\_k, len(unrated\_movies)))

        candidate\_movies = self.cinema.movies[

            ~self.cinema.movies.index.isin(self.cinema.rates.index)

        ].copy()

        if candidate\_movies.empty:

            return candidate\_movies

        def calculate\_score(genres, time\_factor=time\_t):

            if not genres:

                return 0

            base\_score = sum(genre\_probs.get(genre, 0) for genre in genres)

            diversity\_bonus = len(set(genres) & set(genre\_probs.keys())) / len(genres)

            time\_weight = min(time\_t / 5.0, 1.0)

            final\_score = base\_score \* (1 - time\_weight) + diversity\_bonus \* time\_weight

            return final\_score

        candidate\_movies["continuous\_score"] = candidate\_movies["genres"].apply(

            lambda genres: calculate\_score(genres, time\_t)

        )

        max\_score = candidate\_movies["continuous\_score"].max()

        if max\_score > 0:

            candidate\_movies["continuous\_score"] = (

                candidate\_movies["continuous\_score"] / max\_score

            )

        recommendations = (

            candidate\_movies[candidate\_movies["continuous\_score"] > 0.01]

            .sort\_values("continuous\_score", ascending=False)

            .head(top\_k)

        )

        return recommendations.drop(columns=["continuous\_score"], errors="ignore")

    def show\_intensity\_matrix(self):

        """Отображение матрицы интенсивностей с дополнительной информацией"""

Продолжение Листинга В

        if self.intensity\_matrix is None:

            success = self.build\_intensity\_matrix()

            if not success:

                self.cinema.console.print(

                    "[red]Не удалось построить матрицу интенсивностей[/red]"

                )

                return

        table = Table(

            title="Матрица интенсивностей переходов (непрерывная цепь Маркова)"

        )

        table.add\_column("From/To", style="cyan")

        for genre in self.genre\_states:

            table.add\_column(genre, style="green", width=10)

        for i, from\_genre in enumerate(self.genre\_states):

            row = [from\_genre]

            for j, to\_genre in enumerate(self.genre\_states):

                intensity = self.intensity\_matrix[i, j]

                if abs(intensity) > 0.001:

                    if i == j:

                        row.append(f"[red]{intensity:+.3f}[/red]")

                    else:

                        row.append(f"{intensity:+.3f}")

                else:

                    row.append("0.000")

            table.add\_row(\*row)

        self.cinema.console.print(table)

        self.cinema.console.print(

            f"\n[bold]Размер матрицы:[/bold] {len(self.genre\_states)}x{len(self.genre\_states)}"

        )

        self.cinema.console.print(

            f"[bold]Количество жанров:[/bold] {len(self.genre\_states)}"

        )

    def \_auto\_visualize\_markov\_chain(self, time\_t=1.0):

        """Автоматическая визуализация и сохранение графа непрерывной цепи Маркова"""

        os.makedirs("markov", exist\_ok=True)

        if self.intensity\_matrix is None or len(self.genre\_states) == 0:

            return

        G = nx.DiGraph()

        for genre in self.genre\_states:

            G.add\_node(genre)

        edge\_labels = {}

        for i, from\_genre in enumerate(self.genre\_states):

            for j, to\_genre in enumerate(self.genre\_states):

                intensity = self.intensity\_matrix[i, j]

                if intensity > 0.001:

                    G.add\_edge(from\_genre, to\_genre, weight=intensity)

                    edge\_labels[(from\_genre, to\_genre)] = f"{intensity:.2f}"

        plt.figure(figsize=(18, 14), dpi=120)

Продолжение Листинга В

        if len(self.genre\_states) <= 6:

            pos = nx.circular\_layout(G, scale=3.0)

        elif len(self.genre\_states) <= 10:

            pos = nx.circular\_layout(G, scale=3.5)

        else:

            pos = nx.spring\_layout(G, k=4, iterations=150, scale=3)

        node\_size = 8000

        nx.draw\_networkx\_nodes(

            G,

            pos,

            node\_size=node\_size,

            node\_color="lightblue",

            alpha=0.95,

            edgecolors="navy",

            linewidths=3,

            node\_shape="o",

        )

        nx.draw\_networkx\_labels(

            G,

            pos,

            font\_size=16,

            font\_weight="bold",

            font\_family="DejaVu Sans",

            verticalalignment="center",

            horizontalalignment="center",

        )

        if G.edges():

            edge\_weights = [G[u][v]["weight"] for u, v in G.edges()]

            max\_weight = max(edge\_weights) if edge\_weights else 1

            edge\_widths = [1.5 + (w / max\_weight) \* 2.5 for w in edge\_weights]

            edge\_alphas = [0.4 + (w / max\_weight) \* 0.6 for w in edge\_weights]

            edges = nx.draw\_networkx\_edges(

                G,

                pos,

                edge\_color="darkred",

                width=edge\_widths,

                alpha=edge\_alphas,

                arrows=True,

                arrowsize=30,

                arrowstyle="-|>",

                connectionstyle="arc3,rad=0.25",

                min\_source\_margin=20,

                min\_target\_margin=25,

                node\_size=node\_size,

                ax=plt.gca(),

            )

            nx.draw\_networkx\_edge\_labels(

                G,

                pos,

                edge\_labels=edge\_labels,

                font\_size=12,

                font\_weight="bold",

                font\_family="DejaVu Sans",

                label\_pos=0.18,

Окончание Листинга В

                verticalalignment="center",

                horizontalalignment="center",

                bbox=dict(

                    boxstyle="round,pad=0.3",

                    facecolor="white",

                    alpha=0.9,

                    edgecolor="lightgray",

                    linewidth=1.5,

                ),

            )

        title\_info = f"Непрерывная цепь Маркова: интенсивности переходов (t={time\_t})\n"

        title\_info += f"Узлы: {len(G.nodes())} | Переходы: {len(G.edges())} | "

        title\_info += f"Сгенерировано: {datetime.now().strftime('%d.%m.%Y %H:%M')}"

        plt.title(

            title\_info,

            fontsize=18,

            fontweight="bold",

            pad=30,

            loc="center",

            fontfamily="DejaVu Sans",

        )

        plt.axis("off")

        plt.tight\_layout(pad=4.0)

        timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")

        filename = f"markov/continuous\_markov\_chain\_t{time\_t}\_{timestamp}.png"

        plt.savefig(

            filename, dpi=150, bbox\_inches="tight", facecolor="white", edgecolor="none"

        )

        plt.close()