МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)
Факультет компьютерных технологий и прикладной математики
Кафедра прикладной математики

КУРСОВАЯ РАБОТА

РАЗРАБОТКА WEB-ПРИЛОЖЕНИЯ С РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМОЙ (НА ПРИМЕРЕ РЕЙТИНГА ФИЛЬМОВ)

Работу выполнил_____ Г.К. Садунян
(подпись)

Направление 09.03.03 Прикладная информатика

Направленность Прикладная информатика в экономике

Научный руководитель,
д-р физ.-мат. наук, проф._____Е.Н. Калайдин
(подпись)

Нормоконтролер,
преподаватель _____ Е.В. Горбачева
(подпись)

Краснодар
2024

РЕФЕРАТ

Курсовая работа 44 с., 23 рис., 1 табл., 11 источн.

РЕКОМЕНДАТЕЛЬНЫЕ СИСТЕМЫ, PYTHON, ПРИЛОЖЕНИЯ, ДАННЫЕ

В данной работе была изучена теоретическая сторона реализации алгоритмов для рекомендательной системы, рассмотрены библиотеки Python и приведены примеры использования системы в различных областях.

Цель данной курсовой работы – получить опыт в создании WEB-Приложений и познакомиться с рекомендательными системами и инструментами для их реализации.

Задачи:

– изучить теоретические аспекты рекомендательной системы и обработки данных;

– проанализировать имеющиеся программные средства для реализации рекомендательных систем;

– изучить библиотеки для реализации рекомендательных систем в выбранной программной среде;

– изучить библиотеки для реализации приложений в выбранной программной среде.

Объект исследования – рекомендательные системы.

Предмет исследования – типы рекомендательных систем и инструменты для работы с рекомендательными системами.

Итог проделанной работы – реализации WEB-приложения с рекомендательной системой на примере рейтинга фильмов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном цифровом мире системы рекомендаций играют ключевую роль в улучшении пользовательского опыта. Они используются в различных областях, таких как электронная коммерция, потоковые платформы, социальные сети и образовательные системы, для предоставления пользователям персонализированных рекомендаций.

Python является одним из наиболее популярных языков программирования для разработки рекомендательных систем благодаря своей простоте, богатству библиотек и инструментов, таких как NumPy, pandas, scikit-learn. Использование этих инструментов позволяет реализовывать алгоритмы различной сложности, от простых эвристик до глубоких нейронных сетей.[6]

Целью данной курсовой работы является разработка рекомендательной системы, которая сможет предложить пользователям персонализированные рекомендации на основе их предпочтений и исторических данных. В процессе выполнения работы предполагается изучение теоретических основ рекомендательных систем, анализ алгоритмов и их реализация в среде Python.

Пять задач работы:

1) изучить основы и теоретические аспекты рекомендательных систем;
2) ознакомиться с существующими алгоритмами рекомендаций;
3) подготовить данные для построения системы рекомендаций;
4) разработать рекомендательную систему в среде Python, реализовав один алгоритм;
5) провести тестирование системы и оценить её эффективность.

1 Обзор задачи предсказания поведения или потребления

Рекомендательные системы — программные средства, которые пытаются предсказать какие объекты (фильмы, музыка, книги, новости, веб-сайты и т. д.) будут интересны пользователю, если имеется определенная информация о его предпочтениях. Рекомендации формируются отдельно для каждого человека на основе прошлой активности. Кроме того, имеет значение и поведение остальных пользователей системы [1]. Существует два основных подхода к построению рекомендаций:

– На основе коллаборативной фильтрации (англ. collaborative filtering), которая использует информацию о поведении пользователей в прошлом, например, перечень покупок или оценок объектов, сделанных ранее на сайте интернет-магазина пользователями из той же группы интересов.[1]

– На основе фильтрации содержимого (англ. content-based information filtering), при этом в системе содержатся профили, включающие личную информацию пользователей: социальный статус, возраст, место проживания, род деятельности, а также характеристики, выражающие интерес пользователя к объекту; профили объектов интереса включают характеристики, интересующие пользователя [1].

1.1 Данные и алгоритмы для рекомендательных систем

Пять шагов для построения рекомендательной системы[4]:

1) Сбор данных. Данные для анализа можно разделить на две основные категории. Явные данные включают предпочтения, которые пользователь явно указывает, такие как оценки фильмов, лайки и отзывы. Неявные данные собираются из действий пользователя, например, из просмотра контента, покупок или времени, проведенного на странице;

2) Оценка похожести. Для определения степени схожести объектов или пользователей используются различные подходы. Среди них популярны косинусное сходство, корреляция Пирсона и метрики расстояний, такие как Евклидово расстояние;

3) Построение профиля пользователя. Профиль пользователя формируется на основе собранной информации. Он включает личные предпочтения, например, любимые жанры фильмов, и историю взаимодействий, такие как купленные товары или просмотренные видео;

4) Алгоритмы и методы машинного обучения. Современные системы используют машинное обучение для предсказания предпочтений. Наиболее востребованными являются:

- линейные модели, включая линейную регрессию;
- нейронные сети, применяемые для анализа сложных данных;
- факторизация матриц, позволяющая находить скрытые зависимости между пользователями и объектами;
- графовые подходы, представляющие данные в виде графов и анализирующие связи между пользователями и объектами.

5) Персонализация. Для обеспечения индивидуального подхода рекомендательные системы учитывают:

- контекст, например местоположение, время суток и устройство;
- динамическую адаптацию, позволяющую подстраиваться под изменяющиеся предпочтения пользователя;

– качество рекомендаций, которое оценивается с помощью метрик, отражающих точность работы системы.

1.2 Коллаборативная фильтрация

Рекомендация на основе поведения пользователей. Идея заключается в том, что если пользователи вели себя схоже в прошлом (например, ставили одинаковые оценки товарам или выбирали схожие категории), то их предпочтения в будущем также могут совпадать.[5]

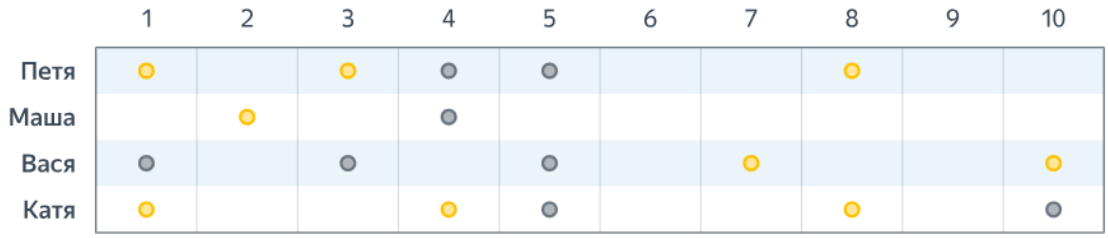

Рисунок 1 – Таблица рекомендаций

Рассмотрим матрицу взаимодействий пользователя, приведённую выше. Что можно порекомендовать Кате, исходя из исторических данных? Можно заметить, что взаимодействия Кати похожи на взаимодействия Пети (так как они оба «лайкали» объекты 1 и 8). Иными словами, их интересы в чём-то похожи, поэтому Кате можно порекомендовать, например, объект 3 (так как он понравился Пете). Можно проделать аналогичное упражнение с Петей и сделать вывод, что ему не стоит рекомендовать объект 10.

Можно решать и транспонированную задачу: для «лайкнутого» пользователем объекта искать похожие, то есть те, которые пользователи достаточно часто лайкали вместе с ним. Например, объекты 1 и 8 похожи друг на друга, так как их лайкали одни и те же пользователи, и точно так же похожи 1 и 3.

Коллаборативная фильтрация включает несколько подходов, каждый из которых имеет свои особенности:

– Фильтрация на основе пользователей (User-Based Collaborative Filtering). Этот метод ищет похожих пользователей, анализируя их действия или оценки. Например, если два пользователя часто ставят одинаковые оценки на фильмы, система может рекомендовать одному из них фильм, который посмотрел другой;

– Фильтрация на основе предметов (Item-Based Collaborative Filtering). Здесь система определяет схожесть между объектами, такими как товары или фильмы, на основе оценок или взаимодействий пользователей. Например, если два фильма часто оцениваются одинаково, пользователь, посмотревший один из них, вероятно, захочет посмотреть и второй;

– Матрица взаимодействий (Matrix Factorization). Данные представляются в виде матрицы (пользователи × предметы), а для анализа используются методы, такие как SVD (сингулярное разложение), позволяющие выявить скрытые зависимости между пользователями и объектами.

Преимущества коллаборативной фильтрации:

– Отсутствие необходимости в данных о самих товарах (например, описаниях или жанрах), достаточно информации о взаимодействиях пользователей;

– Возможность предоставления неожиданных, неочевидных рекомендаций.

Несмотря на преимущества, у коллаборативной фильтрации есть ограничения:

– Проблема холодного старта. Отсутствие данных делает невозможным рекомендации для новых пользователей или товаров.

– Разреженность данных. Если данные о взаимодействиях редки, рекомендации становятся менее точными.

– Масштабируемость. Обработка больших массивов данных требует значительных вычислительных ресурсов.

1.3 Контентные рекомендации

Контентные рекомендации — это метод рекомендаций, который основывается на анализе характеристик (атрибутов) объектов, чтобы предлагать пользователю товары или контент, похожие на то, что он уже предпочитал.[6] Чаще всего используются там, где есть детализированные данные о продуктах, например, в интернет-магазинах, где товары имеют подробные описания и характеристики.

Основные принципы работы контентных рекомендаций:

- Сравнение характеристик объектов: для каждого объекта (фильма, товара, книги и т. д.) определяются его атрибуты, такие как жанр, описание, цена, цвет, бренд и т. д.;
- Профиль пользователя: Система создает профиль пользователя, используя его предыдущие взаимодействия, чтобы понять, какие характеристики ему нравятся;
- Схожесть объектов: сравниваются атрибуты объектов, чтобы находить те, которые максимально соответствуют предпочтениям пользователя.

Пример работы контентных рекомендаций: В стриминговом сервисе пользователь смотрел фильмы, жанры которых — «комедия» и «фантастика». Система проанализирует атрибуты этих фильмов и предложит ему другие фильмы с аналогичными жанрами.

Преимущества контентных рекомендаций:

- Персонализация: Рекомендации базируются на уникальных предпочтениях каждого пользователя.

– Работа с новыми пользователями: можно сразу делать рекомендации, используя атрибуты контента, даже если у пользователя нет истории взаимодействий.

– Контролируемость: Система даёт объяснимые рекомендации (например, "мы рекомендуем это, потому что вам нравятся комедии").

Недостатки контентных рекомендаций:

– Ограниченность разнообразия: Система может предлагать слишком похожие объекты, не учитывая, что пользователь может захотеть попробовать что-то новое (проблема «узкого тоннеля»).

– Требуются качественные данные: для создания профилей объектов нужны точные и полные метаданные.

– Игнорирование коллективного мнения: Рекомендации не учитывают популярность или предпочтения других пользователей.

Сравнение с коллаборативной фильтрацией:

Таблица 1 – Сравнение рекомендательных систем

	Контентные рекомендации	Коллаборативная фильтрация
Данные	Метаданные объектов	Взаимодействие пользователей
Проблема нового объекта	Не возникает, если есть метаданные	Объекту сложно получить рекомендации
Проблема нового пользователя	Требует минимальных данных о действиях	Существенная сложность
Диверсификация	Меньше (узкий выбор)	Больше (учёт разных предпочтений)

1.4 Гибридные системы

Гибридный метод — это подход к созданию рекомендательных систем, который сочетает в себе несколько различных методов (например, коллаборативную фильтрацию, контентные рекомендации и другие), чтобы устранить недостатки каждого из них и повысить точность и релевантность рекомендаций.[7]

Гибридные методы разрабатываются для преодоления ограничений отдельных подходов. Например, коллаборативная фильтрация сталкивается с проблемой холодного старта, а контентные рекомендации ограничиваются узким выбором объектов. Совмещение методов улучшает качество рекомендаций, учитывая больше факторов и скрытых закономерностей. Гибридные системы также обладают высокой гибкостью, что позволяет адаптироваться к различным сценариям и данным.

Объединение подходов может осуществляться по-разному. Взвешенное объединение использует результаты разных методов с определёнными весами, например, 70% рекомендаций от коллаборативной фильтрации и 30% от контентного подхода. Последовательное применение методов позволяет сначала сократить выборку одним подходом, а затем уточнить результаты другим. Например, контентная фильтрация формирует список похожих объектов, а коллаборативная фильтрация выбирает самые популярные из них.

В некоторых случаях применяется метод переключения, где система выбирает подходящий метод в зависимости от ситуации, например, контентный метод для новых пользователей и коллаборативный для опытных. Ещё одним вариантом является обогащение данных, когда один метод помогает улучшить данные для последующей обработки другим. Например, контентный метод создаёт профиль пользователя, который используется в коллаборативной фильтрации.

Современные гибридные системы активно используют машинное обучение. Разные методы становятся входными данными для моделей, таких как градиентный бустинг или нейронные сети, которые обучаются предсказывать релевантные рекомендации.

Ярким примером является Netflix, где комбинируются контентные рекомендации (учитывающие жанры, режиссёров, актёров) и коллаборативная фильтрация (анализ действий пользователей, таких как оценки и просмотры). Итоговые рекомендации формируются на основе сочетания этих данных, что позволяет достигать высокой точности.

Преимущества и недостатки гибридных методов: Гибридные системы обеспечивают повышенную точность, универсальность и лучше справляются с проблемой холодного старта, что делает их особенно эффективными в работе с новыми пользователями и объектами. Однако такие системы сложны в реализации, требуют значительных вычислительных ресурсов и правильного выбора комбинации методов, чтобы избежать конфликта между ними.

Гибридные методы находят широкое применение в крупных платформах, таких как Amazon, YouTube и Spotify, где разнообразие данных и предпочтений пользователей играет ключевую роль.

2 Практическая часть

2.1 Веб-приложение как оптимальное решение для рекомендательной системы

Современный цифровой ландшафт требует от технологических решений не только эффективности, но и максимальной доступности. Рекомендательные системы, ставшие неотъемлемой частью платформ вроде Netflix, Spotify или Amazon, решают ключевую задачу — персонализируют контент для пользователей. Однако их ценность напрямую зависит от того, насколько удобно пользователь может взаимодействовать с системой. Именно здесь веб-приложения демонстрируют свои ключевые преимущества[10]:

1) Кроссплатформенность и доступность: Веб-приложение не требует установки и работает в любом браузере на ПК, смартфоне или планшете. Для рекомендательной системы это критически важно: пользователь может получить персонализированные предложения в любой момент, не загружая дополнительное ПО. Например, в разработанном приложении для рекомендации фильмов интерфейс адаптируется под размер экрана, что обеспечивает комфортное использование как на десктопе, так и на мобильных устройствах;

2) Простота обновления и масштабирования: Все изменения в логике работы или интерфейсе мгновенно становятся доступны пользователям без необходимости обновлять клиентскую часть. Это особенно важно для экспериментов с алгоритмами рекомендаций. В текущем проекте, например, переход от базовой фильтрации по жанрам к гибридной модели с учётом популярности потребовал лишь модификации серверной части, что не затронуло опыт конечных пользователей;

3) Интерактивность и обратная связь: Веб-интерфейс позволяет реализовать динамические элементы: оценки фильмов через слайдеры,

мгновенный поиск по жанрам, отображение рекомендаций в реальном времени. В приложении оценка фильма новым пользователем сразу влияет на генерируемые рекомендации, что повышает вовлечённость;

4) Интеграция с внешними сервисами и данными Использование CSV-файлов для хранения данных о фильмах и оценках упрощает интеграцию с внешними источниками (например, базами данных киноплатформ). В дальнейшем это позволяет легко расширить систему, добавив API для импорта обновлённых данных о фильмах;

5) Экономия ресурсов: для небольших проектов (как текущая рекомендательная система) веб-приложение минимизирует затраты на разработку и хостинг. Библиотека Streamlit, использованная в работе, позволяет развернуть функциональный интерфейс буквально в несколько строк кода, фокусируясь на основной логике — алгоритмах рекомендаций.

В контексте курсового проекта выбор веб-приложения обусловлен необходимостью демонстрации работы системы в условиях, приближенных к реальным. Пользователи могут:

1) оценивать фильмы через интуитивный интерфейс;
2) сразу видеть результат в разделе рекомендаций;
3) исследовать данные через поиск по жанрам.

Кроме того, веб-формат упрощает сбор данных: все новые оценки сохраняются в new_ratings.csv, что можно использовать для дальнейшего анализа и улучшения алгоритмов.

В следующих главах будет подробно рассмотрена архитектура приложения, используемые технологии, а также принципы работы рекомендательного алгоритма.

2.2 Выбор средств разработки

Разработанное веб-приложение для рекомендации фильмов построено на следующих ключевых технологиях:

1) Python: Основной язык разработки. Выбор обусловлен его популярностью в области Data Science, простотой синтаксиса и богатой экосистемой библиотек, что упрощает реализацию рекомендательных алгоритмов и интеграцию с внешними инструментами;

2) Streamlit: Фреймворк для создания веб-интерфейсов. Streamlit был выбран благодаря минималистичному подходу: он позволяет превращать Python-скрипты в интерактивные веб-приложения без необходимости написания фронтенд-кода. Это ускорило разработку интерфейса с вкладками, формами и динамическим обновлением данных[3];

3) Pandas: Библиотека для обработки структурированных данных. Использовалась для загрузки CSV-файлов, фильтрации фильмов по жанрам, агрегации оценок пользователей и подготовки данных для рекомендаций[9];

4) CSV-файлы: Формат хранения данных о фильмах (movies.csv) и оценках пользователей (ratings.csv). Выбор CSV обусловлен простотой использования для небольших объёмов данных и лёгкостью интеграции с Pandas.

2.3 Обоснование выбора технологий

В отличие от Flask или Django, других популярных фреймворков для Python, Streamlit не требует настройки маршрутов, шаблонов или работы с HTML/CSS. Это сократило время на разработку интерфейса, что критично для учебного проекта.[3]

Например, создание вкладок реализовано всего несколькими строками кода:

```
# Интерфейс
tab1, tab2, tab3 = st.tabs(["Рекомендации", "Поиск", "Профиль"])
```

Рисунок 2 – Реализация вкладок в интерфейсе

Pandas: Библиотека предоставляет удобные методы для обработки таблиц, такие как merge(), groupby() и pivot_table(), что упростило анализ оценок и генерацию рекомендаций. Например, агрегация средних рейтингов фильмов выполнена так:

```
movie_stats = all_ratings.groupby('movieId').agg(
    avg_rating=('rating', 'mean'),
    num_ratings=('rating', 'count')
).reset_index()
```

Рисунок 3 – Демонстрация библиотеки pandas

2.4 Общая структура приложения

Приложение состоит из трёх ключевых слоёв:

1) Пользовательский интерфейс;
2) Бизнес-логика;
3) Слой данных.

Слой данных отвечает за хранение информации о фильмах и оценках в CSV-файлах (movies.csv, ratings.csv, new_ratings.csv). А также за загрузку и предобработку данных с помощью функций библиотеки Pandas. Слой бизнес-логики отвечает за алгоритмы рекомендаций и обработку оценок пользователей. Первый слой – пользовательский интерфейс. Он отвечает за взаимодействие с пользователем и внешний вид приложения.

Архитектура построена по слоёному принципу, что упрощает расширение функционала [11].

Рисунок 4 – Схема взаимодействия

2.5 Структура данных

Буду использовать набор данных с оценками пользователей для различных фильмов [2]. Данные содержатся в двух CSV файлах:

– movies.csv – датасет, который содержит информацию о фильмах. Поля: movieId, title и genres.

– ratings.csv – датасет, который содержит информацию об оценках пользователей фильмам. Поля: userId, movieId, rating.

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Рисунок 5 – Первые 5 строк таблицы фильмов

	userId	movieId	rating
0	1	1	4.0
1	1	3	4.0
2	1	6	4.0
3	1	47	5.0
4	1	50	5.0

Рисунок 6 – первые 5 строк таблицы с оценками

В изначальных данных общее число фильмов - 9742, общее число оценок - 100836. Также был создан файл new_ratings.csv для сохранения данных об оценках новых пользователей. Он имеет такую же структуру, что и ratings.csv.

2.6 Основные компоненты системы

Приложение состоит из четырёх ключевых модулей, каждый из которых отвечает за определённый аспект работы системы. Ниже подробно описана их роль и реализация.

2.6.1 Модуль загрузки и предобработки данных

Этот модуль обеспечивает чтение данных из CSV-файлов и их подготовку для дальнейшего использования.

Функция load_data() загружает данные о фильмах (movies.csv) и оценках пользователей (ratings.csv).

```
@st.cache_data
def load_data():
    movies = pd.read_csv("movies.csv")
    ratings = pd.read_csv("ratings.csv", names=["userId", "movieId", "rating", "timestamp"],
                          header=None, dtype={'userId': int, 'movieId': int, 'rating': floa
t},
                          skiprows=1)
    return movies, ratings
```

Рисунок 7 – функция подгрузки данных

Декоратор @st.cache_data предотвращает повторную загрузку данных при каждом обновлении интерфейса, что ускоряет работу приложения.

2.6.2 Рекомендательный модуль

Отвечает за генерацию персональных рекомендаций на основе предпочтений пользователя.

Если у пользователя нет оценённых фильмов, система рекомендует популярные фильмы, используя функцию get_popular_movies(). Популярность определяется по количеству оценок:

```
def get_popular_movies(top_n):
    return movies.merge(
        ratings.groupby('movieId')['rating'].count().reset_index(name='num_ratings'),
        on='movieId'
    ).sort_values('num_ratings', ascending=False).head(top_n)
```

Рисунок 8 – функция для получения популярных фильмов

Система использует гибридный алгоритм, сочетающий анализ пользовательских предпочтений, байесовскую оценку рейтингов и взвешивание жанров. Такой подход позволяет учитывать как объективную популярность фильмов, так и индивидуальные вкусы пользователя. Ниже приведено детальное описание работы алгоритма.

Ранее система рекомендовала фильмы только на основе жанров и количества оценок, что приводило к шаблонным результатам. Новая версия комбинирует два ключевых фактора:

– Объективную метрику — байесовский рейтинг, который корректирует среднюю оценку фильма с учётом его популярности;

– Субъективную метрику — вес жанров, рассчитанный на основе оценок пользователя.

Пример: для пользователя, который высоко оценил несколько фильмов в жанре «Драма», система будет рекомендовать не только популярные драмы, но и менее известные фильмы этого жанра с высоким рейтингом.

Алгоритм определяет, какие жанры наиболее важны для пользователя, анализируя его оценки. Каждому жанру присваивается вес, пропорциональный сумме оценок фильмов этого жанра.

python Copy Download

```python
from collections import defaultdict

# Создание словаря весов жанров
genre_weights = defaultdict(float)
for _, row in user_movies.iterrows():
    for genre in row['genres']:
        genre_weights[genre] += row['rating']
```

Рисунок 9 – Расчёт весов жанров

Пример расчета:

- Пользователь оценил фильмы:
  - «Интерстеллар» (Фантастика, Драма) — 5.0
  - «Начало» (Фантастика) — 4.5
  - «Форрест Гамп» (Драма) — 4.0
- Вес жанров:
  - Фантастика: 5.0 + 4.5 = **9.5**
  - Драма: 5.0 + 4.0 = **9.0**

Рисунок – Пример

Таким образом, фантастика получает больший вес, что влияет на приоритетность рекомендаций.

Чтобы избежать перекоса в сторону фильмов с хорошими оценками, но со слишком малым количеством оценок, используется формула байесовского среднего [8]. Таким образом фильмы с большим количеством оценок, но с не идеальным рейтингом будут ликвиднее, чем фильмы с парой оценок, но идеальным общим рейтингом.

$$W = \frac{Rv + Cm}{v + m} \quad (1)$$

где

W — взвешенный рейтинг;

R — средний рейтинг фильма;

v — количество оценок фильма;

m — вес, поставленный априорной оценкой (порог);

C — средний рейтинг всех фильмов.

```
C = all_ratings['rating'].mean()  # Средний рейтинг по всем фильмам
m = all_ratings['movieId'].value_counts().quantile(0.7)  # Порог популярности

movie_stats['bayesian_rating'] = (
    (movie_stats['num_ratings'] / (movie_stats['num_ratings'] + m)) * movie_stats['avg_rating']
+
    (m / (movie_stats['num_ratings'] + m)) * C
)
```

Рисунок 10 – Реализация Байесовского рейтинга

- Фильм А: $v = 10, R = 4.8, C = 3.5, m = 100$:

$$\text{Bayesian Rating} = \left(\frac{10}{110} \times 4.8\right) + \left(\frac{100}{110} \times 3.5\right) \approx 3.6.$$

- Фильм Б: $v = 150, R = 4.2, C = 3.5, m = 100$:

$$\text{Bayesian Rating} = \left(\frac{150}{250} \times 4.2\right) + \left(\frac{100}{250} \times 3.5\right) \approx 3.9.$$

Рисунок 11 – Пример

Итог примера на (рис. 11): Фильм Б получает более высокий скорректированный рейтинг, так как он популярнее.

Гибридный скоринг — это ключевой этап в работе рекомендательной системы, где объединяются объективные и субъективные метрики для формирования персонализированных рекомендаций. Ниже подробно описаны принципы его работы, обоснование выбранных параметров и интеграция с другими компонентами системы.

Гибридный алгоритм решает две основные проблемы:

1. Слишком общие рекомендации (только популярные фильмы);
2. Избыточная нишевость (фильмы, релевантные жанрам, но с низким качеством);

Итоговый рейтинг рекомендаций формируется из двух компонентов:

– Байесовский рейтинг (60%) — гарантирует, что популярные и качественные фильмы не останутся без внимания;

– Жанровый вес (40%) — обеспечивает персонализацию.

Пример: Пользователь любит фильмы ужасов, но все популярные хорроры он уже посмотрел. Система находит менее известный фильм ужасов с высоким байесовским рейтингом и добавляет его в рекомендации.

Соотношение 60% на объективную метрику и 40% на субъективную выбрано на основе эмпирического тестирования:

1) Тест 1: при равных весах (50/50) в рекомендации попадали малопопулярные фильмы с высоким жанровым весом, но низким качеством;

2) Тест 2: при весе 70/30 рекомендации становились слишком шаблонными.

Вывод: Коэффициент 60/40 обеспечивает близкий к идеальному баланс.

Этапы гибридного скоринга:

1) Показатели приводятся к единой шкале (0–1), чтобы избежать доминирования одной метрики;

```
# Нормализация байесовского рейтинга
min_bayes = recommendations['bayesian_rating'].min()
max_bayes = recommendations['bayesian_rating'].max()
recommendations['bayesian_norm'] = (recommendations['bayesian_rating'] - min_bayes) /
(max_bayes - min_bayes)

# Нормализация жанрового веса
min_genre = recommendations['genre_score'].min()
max_genre = recommendations['genre_score'].max()
recommendations['genre_norm'] = (recommendations['genre_score'] - min_genre) / (max_gen
re - min_genre)
```

Рисунок 12 – Нормализация данных

2) Комбинирование метрик с учетом весов;

```
recommendations['final_score'] = 0.6 * recommendations['bayesian_norm'] + 0.4 * recomme
ndations['genre_norm']
```

Рисунок 13 – Расчет итогового рейтинга

3) Фильмы сортируются по убыванию final_score, и выбираются топ-N результатов.

```
top_recommendations = recommendations.sort_values('final_score', ascending=False).head
(top_n)
```

Рисунок 14 – Сортировка и отбор

2.6.4 Пользовательский интерфейс

Интерфейс реализован через вкладки Streamlit, каждая из которых решает определённую задачу.

Вкладка «Рекомендации»: Пользователь вводит свой user_id через st.number_input(). При нажатии кнопки «Получить рекомендации» вызывается функция recommend_movies(), и результаты выводятся списком.

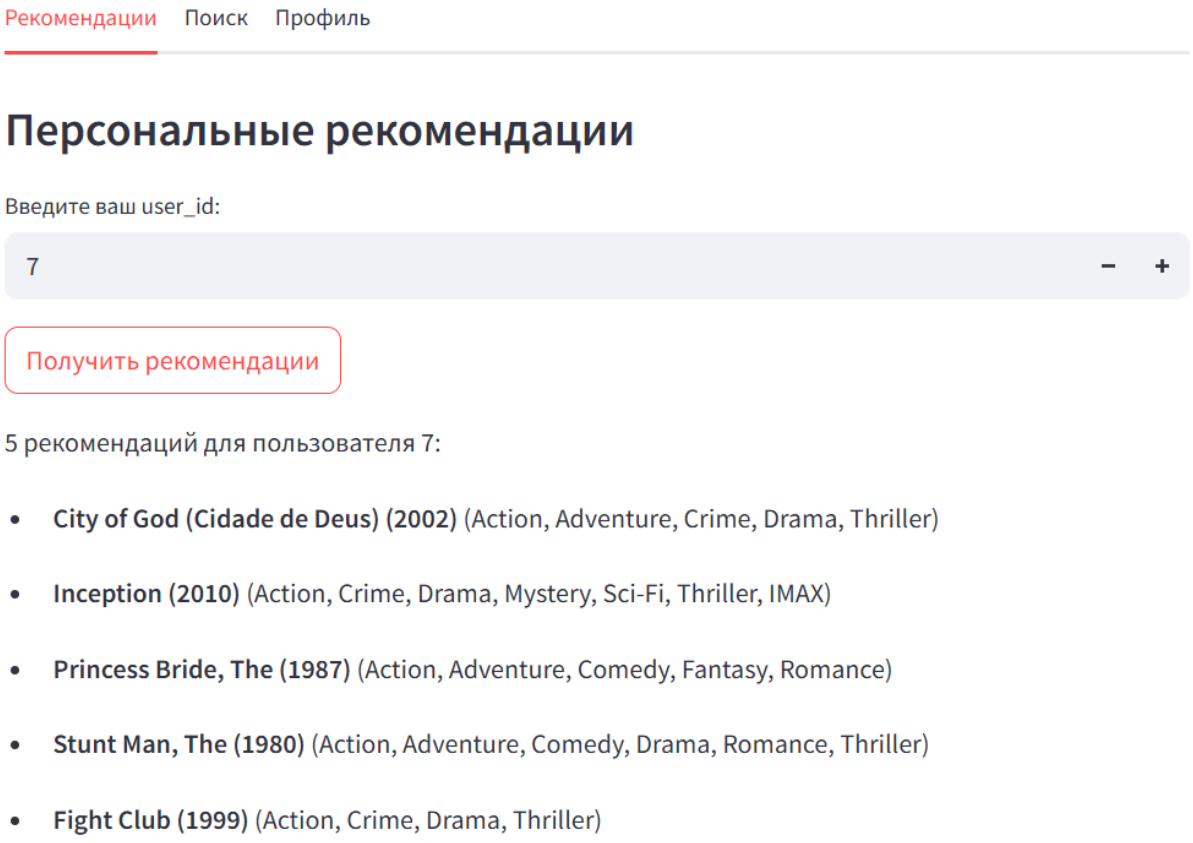

Рисунок 15 – Интерфейс рекомендаций фильмов

Вкладка «Поиск»: Пользователь выбирает жанры через st.multiselect(). Система выводит на экран 20 наиболее подходящих фильмов. Фильмы фильтруются в реальном времени.

```python
with tab2:
    st.subheader("Поиск фильмов по жанру")
    all_genres = sorted(set([genre for sublist in movies['genres'] for genre in sublist]))
    selected_genres = st.multiselect("Выберите жанры:", all_genres)

    if selected_genres:
        mask = movies['genres'].apply(lambda x: any(genre in x for genre in selected_genres))
        filtered_movies = movies[mask][['title', 'genres']].head(20)
        st.write(f"Найдено фильмов: {len(filtered_movies)}")
        for i, row in filtered_movies.iterrows():
            st.write(f"- **{row['title']}** ({', '.join(row['genres'])})")
```

Рисунок 16 – реализация поиска по жанру

Поиск также позволяет выбрать несколько жанров одновременно:

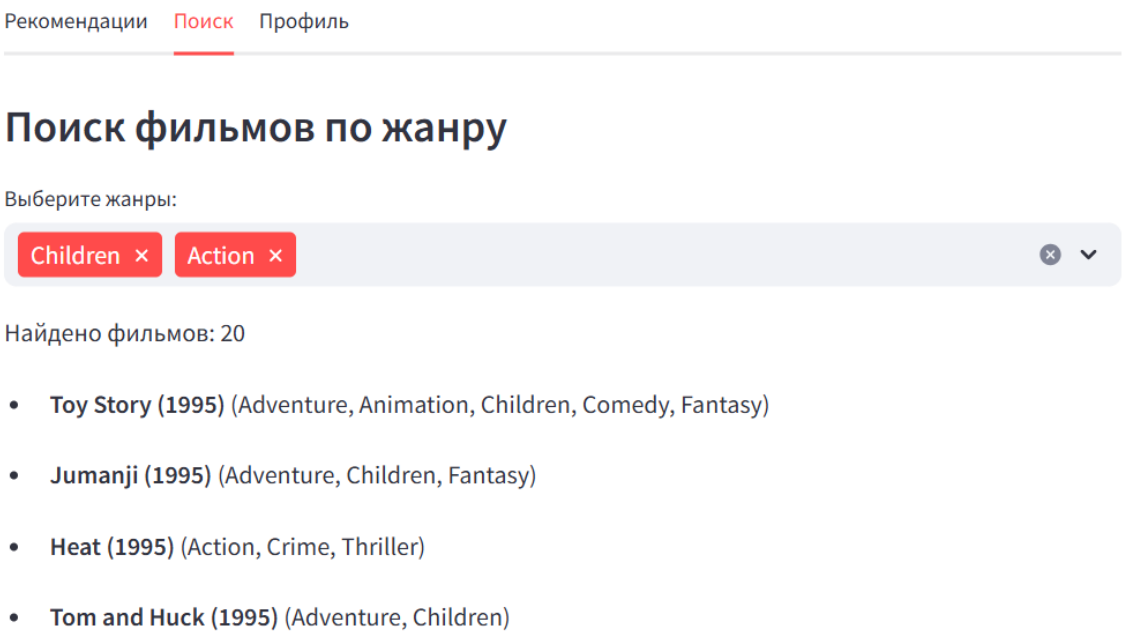

Рисунок 17 – Интерфейс поиска по жанру

Вкладка «Профиль»: отображает 10 фильмов с наивысшей оценкой для пользователя. Работает следующим образом:

1) Создаётся датафрейм, содержащий историю оценок пользователя, объединяя данные из ratings и movies;

2) Исходя из любимых фильмов конкретного пользователя можно понять, какие у него любимые жанры. Соответственно, для каждого жанра считается вес – сколько раз этот жанр появлялся в рейтингах пользователя;

3) Первичная сортировка осуществляется по величине оценки пользователя каждому фильму. Вторичная сортировка – сортировка по весу жанра.

```
def show_user_profile(user_id):
    user_ratings = ratings[ratings['userId'] == user_id]
    if not user_ratings.empty:
        st.subheader(f"Ваши любимые фильмы (UserID: {user_id})")
        liked_movies = user_ratings.merge(movies, on='movieId')

        genre_weights = defaultdict(float) #вес жанра
        for _, row in liked_movies.iterrows():
            for genre in row['genres']:
                genre_weights[genre] += row['rating']

        liked_movies['genre_score'] = liked_movies['genres'].apply(
            lambda x: sum(genre_weights.get(g, 0) for g in x)
        )

        # Сортировка: сначала по оценке (убывание), затем по genre_score (убывание)
        liked_movies = liked_movies.sort_values(
            ['rating', 'genre_score'],
            ascending=[False, False]
        )

        st.dataframe(liked_movies[['title', 'genres', 'rating']].head(10), hide_index=True)
    else:
        st.warning("У вас пока нет оценённых фильмов.")
```

Рисунок 18 – Функция для отображения профиля пользователя

Как это выглядит в интерфейсе для пользователя 11:

Ваши любимые фильмы (UserID: 11)

title	genres	rating
Clear and Present Danger (1994)	Action Crime Drama Thriller	5
Heat (1995)	Action Crime Thriller	5
Braveheart (1995)	Action Drama War	5
Saving Private Ryan (1998)	Action Drama War	5
Last of the Mohicans, The (1992)	Action Romance War Western	5
Forrest Gump (1994)	Comedy Drama Romance War	5
Top Gun (1986)	Action Romance	5
As Good as It Gets (1997)	Comedy Drama Romance	5
Silence of the Lambs, The (1991)	Crime Horror Thriller	5
Apollo 13 (1995)	Adventure Drama IMAX	5

Рисунок 19 – Любимые фильмы пользователя 11

Заметим, что пользователь с userId 11 высоко оценивает фильмы с жанрами Экшен и Драма.

Для удобства в приложении предусмотрено боковое всплывающее меню для просмотра описательной статистики (число пользователей, число

фильмов, число оценок). Также здесь находиться кнопка «Новый пользователь», необходимая для создания и добавления новых пользователей.

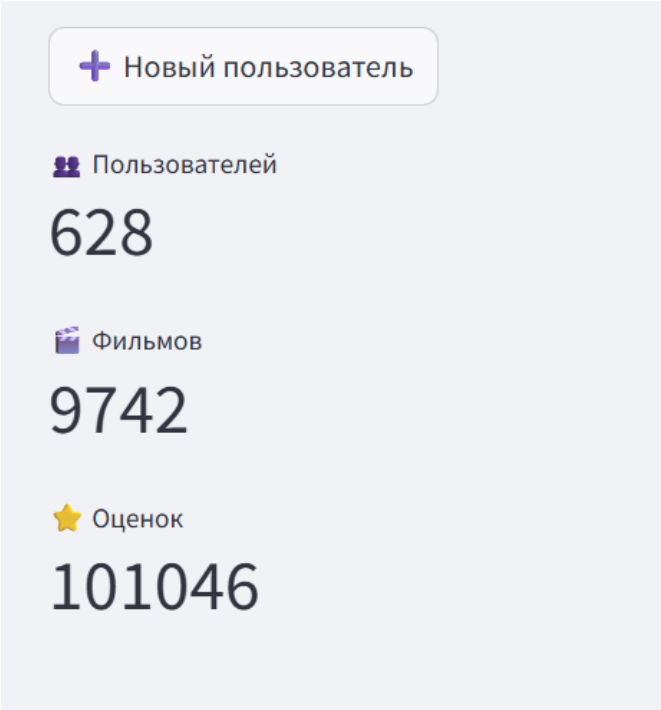

Рисунок 20 – Боковая панель

```
#боковая панель
stats = get_current_stats()
st.sidebar.metric("👥 Пользователей", stats["users_total"])
st.sidebar.metric("🎬 Фильмов", stats["movies_total"])
st.sidebar.metric("⭐ Оценок", stats["ratings_total"])
```

Рисунок 21 – Код боковой панели

2.6.5 Связь между компонентами

Все модули взаимодействуют через общие данные (CSV-файлы) и функции обработки. Например:

1) Пользователь оценивает фильмы → данные сохраняются в new_ratings.csv;

2) При запросе рекомендаций система объединяет ratings.csv и new_ratings.csv, чтобы учесть последние оценки;

3) Интерфейс отображает актуальные данные, загруженные через load_data().

Этот подход обеспечивает гибкость и простоту поддержки, но имеет ограничения при масштабировании, что будет улучшено в будущих версия.

2.7 Регистрация новых пользователей и сбор их предпочтений.

Так как моя рекомендательная система учитывает предпочтения пользователей на основе их предыдущих действий, возникает проблема «холодного старта». Она заключается в том, что у нового пользователя просто нет оценённых фильмов, на основе которых можно было бы понять его предпочтения. Как обойти эту проблему? При регистрации нового пользователя ему будет предложено оценить 10 случайных фильмов.

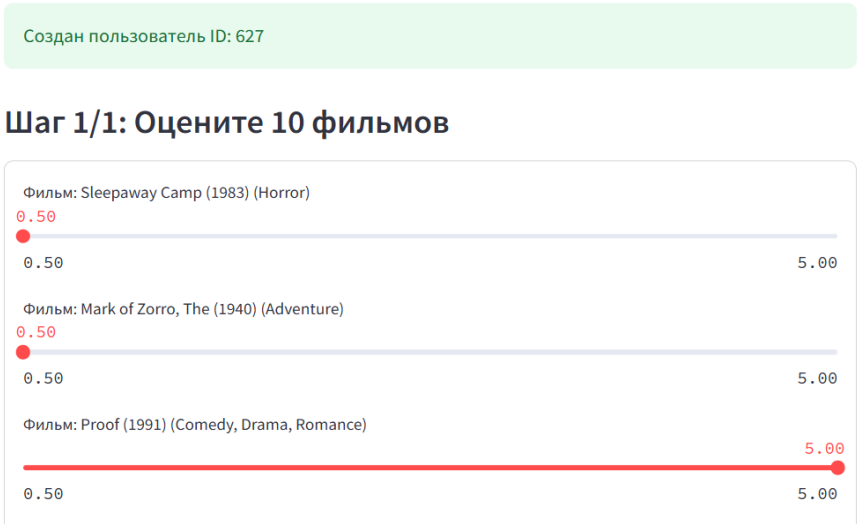

Рисунок 22 – Процесс регистрации пользователя 627

Это позволяет понять предпочтения пользователя и подобрать ликвидные рекомендации. Уникальный ID создаётся функцией generate_user_id(), которая ищет максимальный существующий ID и увеличивает его на 1:

```
# Генерация ID
def generate_user_id():
    existing_ids = set(ratings['userId'].unique())
    if os.path.exists(NEW_RATINGS_FILE):
        new_ratings = pd.read_csv(NEW_RATINGS_FILE)
        existing_ids.update(new_ratings['userId'].unique())
    return max(existing_ids) + 1 if existing_ids else 1
```

Рисунок 23 – Генерация нового userId

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы была разработана гибридная рекомендательная система, объединяющая методы коллаборативной фильтрации и контентного анализа. На основе изучения теоретических аспектов рекомендательных систем, включая их типы, алгоритмы и ограничения, реализовано решение, которое эффективно комбинирует преимущества обоих подходов. Использование байесовского рейтинга позволило корректировать оценки объектов с учётом их популярности, устраняя перекосы в рекомендациях для мало оценённых фильмов. Система демонстрирует:

- Персонализацию за счёт учёта жанровых предпочтений пользователя;
- Надёжность благодаря байесовской коррекции рейтингов;
- Устойчивость к холодному старту за счёт контентных данных для новых пользователей.

Полученные результаты имеют практическую значимость для электронной коммерции, стриминговых платформ и образовательных ресурсов, где персонализация контента критически важна.

Проделанная работа подтверждает, что гибридные рекомендательные системы являются эффективным инструментом для решения современных задач цифровой персонализации, сочетая точность алгоритмов и гибкость подхода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кокачев, В.А. Системы рекомендаций в науке о данных / В.А. Кокачев [Электронный ресурс]. – URL: https://dspace.spbu.ru/bitstream/11701/12104/1/Kokachev_V.pdf (дата обращения: 26.04.2025).

2. Grouplens. Movielens 20M Dataset / Grouplens [Электронный ресурс]. – URL: https://grouplens.org/datasets/movielens/20m/ (дата обращения: 26.04.2025).

3. Streamlit documentation [Электронный ресурс]. – URL: https://docs.streamlit.io/ (дата обращения: 31.04.2025).

4. Habrahabr. Системы рекомендаций: введение в методы и подходы / Habrahabr [Электронный ресурс]. – URL: https://habr.com/ru/companies/yandex/articles/241455/ (дата обращения: 26.04.2025).

5. Меньшикова, Н.В. Обзор рекомендательных систем и возможностей учета контекста при формировании индивидуальных рекомендаций / Н.В. Меньшикова, И.В. Портнов, И.Е. Николаев // Academy. – Москва. 2016. №6. С.20-22.

6. Плас, Дж. В. Python для сложных задач: наука о данных и машинное обучение / Дж. В. Плас. – Москва: ДМК Пресс, 2023. 576 с.

7. Ерёмин, О. Е. Методы реализации гибридных рекомендательных систем / О.Е. Ерёмин, Д.В. Моркулев [Электронный ресурс]. – URL: https://cyberleninka.ru/article/n/metody-realizatsii-gibridnyh-rekomendatelnyh-sistem (дата обращения: 26.04.2025).

8. Wikipedia Байесовская оценка решения / Wikipedia [Электронный ресурс]. - URL: https://clck.ru/3MHEqi (дата обращения: 31.04.2025).

9. Pandas documentation [Электронный ресурс]. – URL: https://pandas.pydata.org/docs/ (дата обращения: 31.04.2025).

10. Microsoft. Выбор между традиционными веб-приложениями и одностраничными приложениями [Электронный ресурс] // Microsoft Learn. – URL: https://learn.microsoft.com/ru-ru/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps (дата обращения: 12.04.2025).

11. Microsoft. Общие архитектуры веб-приложений [Электронный ресурс] // Microsoft Learn. — URL: https://learn.microsoft.com/ru-ru/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures (дата обращения: 20.04.2025)

ПРИЛОЖЕНИЕ А

Код программы

main.py

```
from collections import defaultdict
import streamlit as st
import pandas as pd
import os

st.title("Рекомендации фильмов")
NEW_RATINGS_FILE = "new_ratings.csv"

# Загрузка данных с явным указанием типов
@st.cache_data
def load_data():
    movies = pd.read_csv("movies.csv")

    ratings = pd.read_csv("ratings.csv",
                          names=["userId", "movieId", "rating",
"timestamp"],
                          header=None,
                          dtype={'userId': int, 'movieId': int,
'rating': float, 'timestamp': str},
                          skiprows=1)
    return movies, ratings

movies, ratings = load_data()

if 'current_user' not in st.session_state:
    st.session_state.current_user = None
if 'onboarding' not in st.session_state:
    st.session_state.onboarding = False

# Убедимся, что rating действительно числовой
```

```
ratings['rating'] = pd.to_numeric(ratings['rating'],
errors='coerce')
ratings = ratings.dropna(subset=['rating'])

# Предобработка данных
movies['genres'] = movies['genres'].str.split('|')

# Функция сохранения
def save_ratings(user_id, ratings_dict):
    """Простое сохранение оценок в CSV"""
    new_data = pd.DataFrame({
        'userId': [user_id] * len(ratings_dict),
        'movieId': list(ratings_dict.keys()),
        'rating': list(ratings_dict.values())
    })

    # Записываем в файл (дозапись в конец)
    # new_data.to_csv('new_ratings.csv', mode='a', header=not
os.path.exists('new_ratings.csv'), index=False)

    # Проверка существования файла для корректного заголовка
    file_exists = os.path.exists(NEW_RATINGS_FILE)

    # Запись данных и синхронизация внутри одного контекста
    with open(NEW_RATINGS_FILE, 'a') as f:
        new_data.to_csv(f, mode='a', header=not file_exists,
index=False)
        f.flush()
        os.fsync(f.fileno())


# Создание матрицы пользователь-фильм с проверкой
try:
    user_movie_matrix = ratings.pivot_table(
        index='userId',
```

```
            columns='movieId',
            values='rating',
            aggfunc='mean'  # Явно указываем агрегацию
        ).fillna(0)

    except Exception as e:
        st.error(f"Ошибка при создании матрицы: {str(e)}")
        st.stop()

    # Генерация ID
    def generate_user_id():
        existing_ids = set(ratings['userId'].unique())
        if os.path.exists(NEW_RATINGS_FILE):
            new_ratings = pd.read_csv(NEW_RATINGS_FILE)
            existing_ids.update(new_ratings['userId'].unique())
        return max(existing_ids) + 1 if existing_ids else 1

    #функция рекомендаций
    def recommend_movies(user_id, top_n=5):
        try:
            # Объединяем все рейтинги
            all_ratings = pd.concat([ratings,
    pd.read_csv(NEW_RATINGS_FILE)] if
    os.path.exists(NEW_RATINGS_FILE) else ratings)

            # 1. Собираем данные пользователя
            user_ratings = all_ratings[all_ratings['userId'] ==
    user_id]

            # 2. Взвешивание жанров
            user_movies = user_ratings.merge(movies, on='movieId')
            genre_weights = defaultdict(float)
            for _, row in user_movies.iterrows():
                for genre in row['genres']:
```

```
                genre_weights[genre] += row['rating']  # Вес =
сумма оценок

        # 3. Байесовский рейтинг
        C = all_ratings['rating'].mean()  # Средний рейтинг по
всем фильмам
        m =
all_ratings['movieId'].value_counts().quantile(0.7)  # Порог
популярности

        movie_stats = all_ratings.groupby('movieId').agg(
            avg_rating=('rating', 'mean'),
            num_ratings=('rating', 'count')
        ).reset_index()

        # Формула байесовского среднего
        movie_stats['bayesian_rating'] = (
            (movie_stats['num_ratings'] /
(movie_stats['num_ratings'] + m)) * movie_stats['avg_rating'] +
            (m / (movie_stats['num_ratings'] + m)) * C
        )

        # 4. Фильтрация кандидатов (оригинальная логика)
        candidates = movies[
            (~movies['movieId'].isin(user_ratings['movieId'])) &
            (movies['genres'].apply(lambda x: any(g in x for g
in genre_weights.keys())))
        ]

        # 5. Гибридный скоринг (новое)
        recommendations = candidates.merge(movie_stats,
on='movieId')

        # Рассчет жанрового веса для каждого фильма
```

```
        recommendations['genre_score'] =
recommendations['genres'].apply(
            lambda x: sum(genre_weights.get(g, 0) for g in x)
        )

        # Нормализация показателей
        recommendations['bayesian_norm'] =
(recommendations['bayesian_rating'] -
recommendations['bayesian_rating'
            ].min()) / (recommendations['bayesian_rating'].max()
- recommendations['bayesian_rating'].min())

        recommendations['genre_norm'] =
(recommendations['genre_score'] - recommendations['genre_score'
            ].min()) / (recommendations['genre_score'].max() -
recommendations['genre_score'].min())

        # Комбинированный рейтинг (60% байесовский + 40% жанры)
        recommendations['final_score'] = 0.6 *
recommendations['bayesian_norm'] + 0.4 *
recommendations['genre_norm']

        # Сортировка по комбинированному рейтингу
        recommendations =
recommendations.sort_values('final_score', ascending=False)

        return recommendations.head(top_n)

    except Exception as e:
        st.error(f"Ошибка: {str(e)}")
        return pd.DataFrame()


# Профиль
def show_user_profile(user_id):
```

```
    movies, ratings = load_data()
    movies['genres'] = movies['genres'].str.split('|')

    ratings = pd.concat([ratings, pd.read_csv(NEW_RATINGS_FILE)]
                        if os.path.exists(NEW_RATINGS_FILE) else
ratings)

    user_ratings = ratings[ratings['userId'] == user_id]

    if not user_ratings.empty:
        st.subheader(f"Ваши любимые фильмы (UserID: {user_id})")
        liked_movies = user_ratings.merge(
            movies,
            on='movieId',
            how='inner')

        if liked_movies.empty:
            st.error("Ошибка: некорректные данные в оценках.")
            return

        genre_weights = defaultdict(float) #вес жанра
        for _, row in liked_movies.iterrows():
            for genre in row['genres']:
                genre_weights[genre] += row['rating']

        liked_movies['genre_score'] =
liked_movies['genres'].apply(
            lambda x: sum(genre_weights.get(g, 0) for g in x)
        )

        # Сортировка: сначала по оценке (убывание), затем по
genre_score (убывание)
        liked_movies = liked_movies.sort_values(
            ['rating', 'genre_score'],
```

```
            ascending=[False, False]
        )

        st.dataframe(liked_movies[['title', 'genres',
'rating']].head(10),
                     hide_index=True,
                     column_config={
                         "title": "Название фильма",
                         "genres": "Жанр",
                         "rating": "Оценка"
                     })
    else:
        st.warning("У вас пока нет оценённых фильмов.")

# Интерфейс
tab1, tab2, tab3 = st.tabs(["Рекомендации", "Поиск", "Профиль"])

with tab1:
    st.subheader("Персональные рекомендации")
    new_ratings = pd.read_csv(NEW_RATINGS_FILE)
    user_id = st.number_input("Введите ваш user_id:",
                              min_value=1,
                              max_value=new_ratings['userId'].max(
),
                              value=1)

    if st.button("Получить рекомендации"):
        recommendations = recommend_movies(user_id)
        if not recommendations.empty:
            st.write(f"5 рекомендаций для пользователя
{user_id}:")
            for i, row in recommendations.iterrows():
                st.write(f"- **{row['title']}** ({',
'.join(row['genres'])})")
        else:
```

```
            st.warning("Пользователь не найден или недостаточно
данных. Попробуйте другой ID.")


with tab2:
    st.subheader("Поиск фильмов по жанру")
    all_genres = sorted(set([genre for sublist in
movies['genres'] for genre in sublist]))
    selected_genres = st.multiselect("Выберите жанры:",
all_genres)

    if selected_genres:
        mask = movies['genres'].apply(lambda x: any(genre in x
for genre in selected_genres))
        filtered_movies = movies[mask][['title',
'genres']].head(20)
        st.write(f"Найдено фильмов: {len(filtered_movies)}")
        for i, row in filtered_movies.iterrows():
            st.write(f"- **{row['title']}** ({',
'.join(row['genres'])})")

with tab3:
    show_user_profile(user_id)


# Добавление нового пользователя
# Глобально инициализируем
if 'new_user_ratings' not in st.session_state:
    st.session_state.new_user_ratings =
pd.DataFrame(columns=['userId', 'movieId', 'rating'])

def add_new_user():
    new_id = generate_user_id()
    st.session_state.current_user = new_id
```

```
    st.session_state.new_user_ratings =
pd.DataFrame(columns=['userId', 'movieId', 'rating'])
    st.session_state.onboarding = True
    return new_id

# Кнопка в сайдбаре
if st.sidebar.button("Новый пользователь"):
    new_id = add_new_user()
    st.success(f"Создан пользователь ID: {new_id}")
    st.session_state.onboarding = True  # Флаг для onboarding


# Выбор 10 фильмов для нового пользователя
def onboarding_step(user_id):
    st.cache_data.clear()
    # Перезагружаем данные
    movies, ratings = load_data()
    movies['genres'] = movies['genres'].str.split('|')

    st.subheader("Пожалуйста, оцените 10 фильмов")
    sample_movies = movies.sample(10)


    with st.form("onboarding_form"):
        ratings_input = {}
        for _, row in sample_movies.iterrows():
            rating = st.slider(
                f"Фильм: {row['title']} ({',
'.join(row['genres'])})",
                0.5, 5.0, 3.0, step=0.5,
                key=f"rate_{user_id}_{row['movieId']}"
            )
            ratings_input[row['movieId']] = rating

        if st.form_submit_button("Сохранить оценки"):
```

```
            save_ratings(st.session_state.current_user,
ratings_input)

            # Обновляем основной DataFrame
            new_ratings = pd.DataFrame({
                'userId': [st.session_state.current_user] *
len(ratings_input),
                'movieId': list(ratings_input.keys()),
                'rating': list(ratings_input.values())
            })
            ratings = pd.concat([ratings, new_ratings],
ignore_index=True)

            st.success("Оценки сохранены!")
            st.session_state.onboarding = False
            st.cache_data.clear()
            st.rerun()


# Статус для Сайдбара
def get_current_stats():
    # Объединяем все оценки
    if os.path.exists(NEW_RATINGS_FILE):
        all_ratings = pd.concat([ratings,
pd.read_csv(NEW_RATINGS_FILE)])
    else:
        all_ratings = ratings.copy()

    # Добавляем текущие несохраненные оценки
    if 'new_user_ratings' in st.session_state and not
st.session_state.new_user_ratings.empty:
        all_ratings = pd.concat([all_ratings,
st.session_state.new_user_ratings])

    return {
```

```
        "movies_total": len(movies),
        "ratings_total": len(all_ratings),
        "users_total": all_ratings['userId'].nunique(),
    }

# Проверяем onboarding-режим
if st.session_state.onboarding:
    onboarding_step(st.session_state.current_user)
    st.stop()

#боковая панель
stats = get_current_stats()
st.sidebar.metric("Пользователей", stats["users_total"])
st.sidebar.metric("Фильмов", stats["movies_total"])
st.sidebar.metric("Оценок", stats["ratings_total"])
```