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 интеллектуальных информационных систем

КУРСОВАЯ РАБОТА

АНАЛИЗ АЛГОРИТМОВ ДЛЯ СИСТЕМ РЕКОМЕНДАЦИЙ

Работу выполнил _____ А. Р. Чениб
(подпись)

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

Направленность Технология программирования

Научный руководитель
канд. экон. наук, доц. _____ А. В. Коваленко
(подпись)

Нормоконтролер
канд. физ.-мат. наук, доц. _____ Г. В. Калайдина
(подпись)

Краснодар
2020

РЕФЕРАТ

Курсовая работа 29 с., 14 рис., 12 источников.

РЕКОМЕНДАТЕЛЬНЫЕ СИСТЕМЫ, РЕКОМЕНДАЦИИ, КОЛЛАБОРАТИВНАЯ ФИЛЬТРАЦИЯ, МАТРИЦА ОЦЕНОК, СКРЫТЫЕ ФАКТОРЫ, РАЗЛОЖЕНИЕ МАТРИЦЫ, ГРАДИЕНТНЫЙ СПУСК

Объектом исследования являются системы рекомендаций, особенности их практической реализации.

Цель курсовой работы – изучение методов и практических аспектов создания систем рекомендаций. Итог проделанной работы – создание приложения для составления списка рекомендуемых фильмов.

В результате на языке программирования Python была написана программа, выполняющие следующие функции:

– получение матрицы оценок пользователей;

– реализация алгоритма сингулярного разложения матрицы;

– реализация алгоритма стохастического градиентного спуска;

– вывод фильмов с наивысшим рейтингом в виде ранжированного списка.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С ростом объема информации в Интернете электронные ресурсы сталкиваются с тем, что пользователи в большинстве случаев видят небольшую часть содержимого ресурса. Например, у интернет-магазина некоторый продукт может иметь большой спрос, в то время как сопутствующие ему товары могут быть не замечены покупателями. В связи с подобными ситуациями в электронной торговле необходимы оперативная классификация продукции и анализ поведения пользователей.

В настоящее время задача адаптации предоставляемого контента востребована не только в системах электронной коммерции, но и во многих других сферах. Так, своего рода подбор объектов на основе поведения пользователя реализован виде списка популярных программ на электронных устройствах (например, предложения Siri на ОС iOS), предложениях слов из словаря при вводе с клавиатуры.

Острую необходимость в подобных системах испытывают так называемые контент-провайдеры – ресурсы, предоставляющие различный контент. К таковым можно отнести социальные сети, новостные агрегаторы, агрегаторы товаров, платформы отзывов, такие как, например, Imhonet, онлайн-кинотеатры и музыкальные сервисы.

Специфика каждого сервиса определяет свои способы решения выше изложенных проблем. И основная задача при построении систем рекомендаций – подобрать подходящие технологии для учета поведения пользователя и подбора содержимого.

В данной курсовой работе будет продолжено рассмотрение основных понятий, используемых в разработке рекомендательных систем. Будут описаны и реализованы на языке программирования Python алгоритмы коллаборативной фильтрации с использованием понижения размерности матрицы, а именно метод сингулярного разложения матрицы и метод стохастического градиентного спуска.

1 Основные сведения о рекомендательных системах

1.1 Историческое развитие систем рекомендаций

История развития и практического использования рекомендательных систем имеет сравнительно небольшой возраст. Однако их теоретические основы были заложены еще в середине прошлого века. Так, в 50-е годы в области машинного обучения были сформулированы математические подходы и описаны модели самообучающихся алгоритмов, которые до сих пор лежат в основе всех реализаций рекомендательных систем.

Первые программные средства для рекомендаций появились в начале 90-х годов [1], когда коллаборативная фильтрация стала применяться как решение для борьбы с избыточной информацией в Интернете. Считается, что первая система фильтрации Tapestry была внедрена Дэвидом Голдбергом в компании Xerox Palo Alto Research Center в 1992 году. Ее целью была фильтрация корпоративной почты. Программа позволяла запрашивать информацию о поведении других пользователей: например, показать все сообщения отправленные некоторым сотрудником.

Позже системы фильтрации получили возможность автоматически отслеживать предпочтения похожих пользователей и предлагать объекты, выбранные одним из них, другому. Данный подход использовался лабораторией GroupLens университета Миннесоты для нахождения статей в сети Usenet, которые могли бы заинтересовать конкретного пользователя. Пользователям предлагалось оценить статьи, а система, в свою очередь, объединяла их с оценками других пользователей для предоставления персонализированных результатов.

В то же время рекомендательными системами стали интересоваться исследователи в области машинного обучения и информационного поиска. Были созданы системы рекомендаций в таких областях, как музыка (система Ringo), фильмы (BellCore Video Recommender) и анекдоты (Jester). Вскоре,

вслед за компьютерными науками, рекомендательные системы стали рассматриваться в маркетинговых исследованиях как инструмент увеличения продаж.

В конце 90-х начали внедряться первые коммерческие системы рекомендаций. Так, Amazon стал советовать товары, смежные с рассматриваемым в данный момент продуктом.

Повышенный интерес к коллаборативной фильтрации привел к появлению компаний, занимающихся внедрением технологий рекомендаций в онлайн-магазины. Создавались новые методы подбора продуктов, включая системы на основе контента и гибридные системы.

Дополнительное внимание рекомендательные алгоритмы привлекли к себе в 2006 году, когда компания Netflix запустила соревнование NetflixPrize. Его целью было создание алгоритма рекомендаций, который смог бы улучшить результат действующего алгоритма CineMatch на 10%. Это вызвало всплеск активности, как в любительских, так и в академических кругах. Объявленная победителям премия в размере одного миллиона долларов демонстрирует ценность рекомендательных алгоритмов для крупных компаний.

В 2000-е годы произошел бурный рост социальных сетей. Facebook одной из первых внедрила алгоритм рекомендаций потенциальных социальных связей. Такие рекомендации несут несколько иные цели, чем рекомендации продукта: социальные сети в значительной степени зависят от количества социальных связей для увеличения доходов от рекламы, поэтому рекомендации потенциальных друзей обеспечивают быстрый рост и связность сети.

В настоящее время развитие рекомендательных систем направлено в сторону применения нейронных сетей. Например, они используются социальными сетями Facebook [2] и Instagram [3]. Также для повышения точности рекомендаций в системах используется учет контекста. Так, платформа Яндекс.Атом позволяет интернет-сайтам подстраивать свои

каталоги, сообщая им, как изменить страницу, чтобы информация для зашедшего пользователя была релевантна [4]. Такая система реализует выдачу рекомендаций между различными сайтами.

1.2 Классификация систем рекомендаций

Рассмотрим основные модели, используемые в рекомендательных системах. Они представлены на рисунке 1:

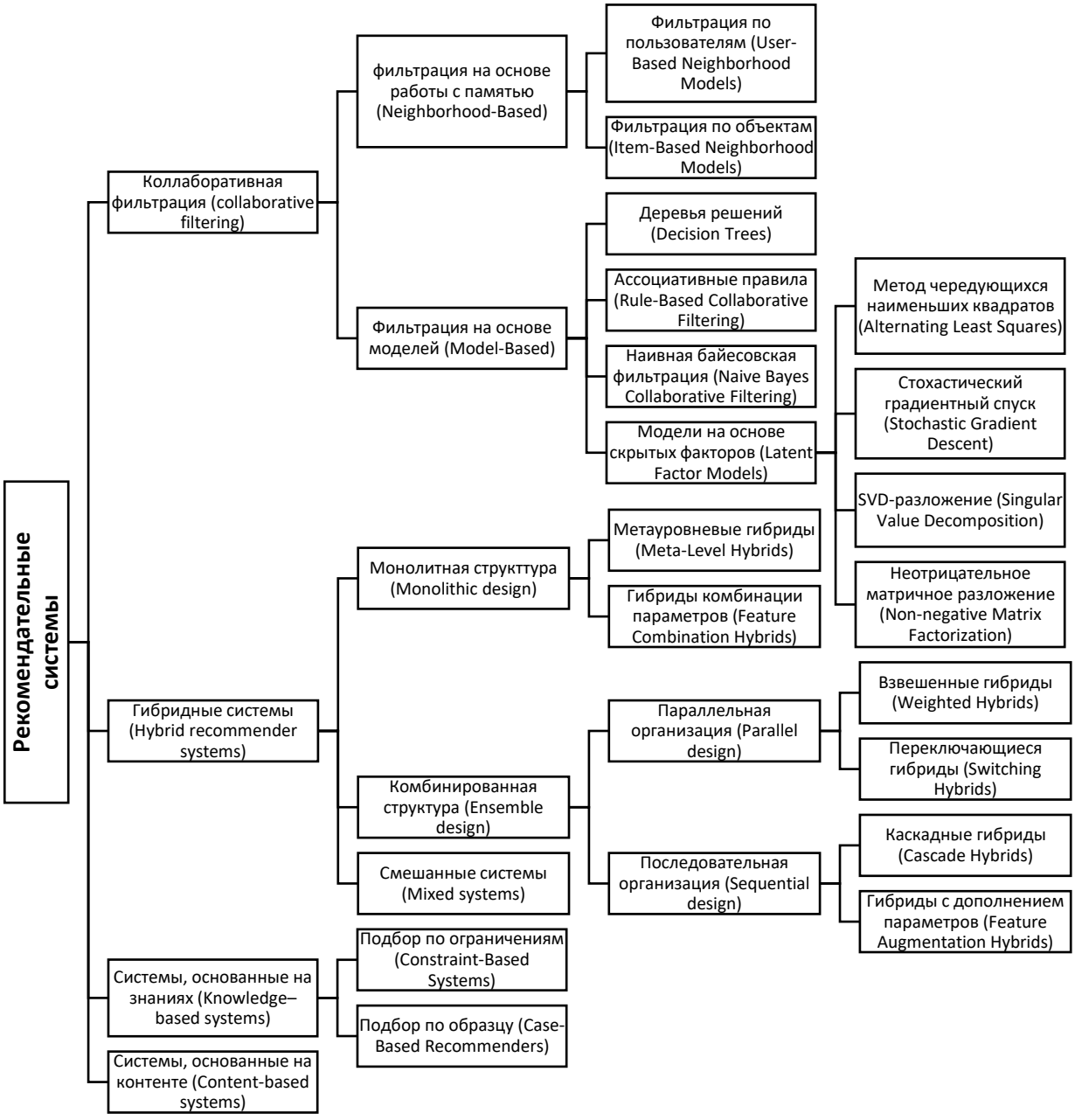

Рисунок 1 – Дерево классификации рекомендательных систем

Выбор модели зависит от различных признаков [5], в частности, от предметной области, имеющихся данных о пользователях и объектах и видов обратной связи от пользователей (явной и неявной). Ниже приводится краткое описание основных подходов к составлению рекомендаций [6].

1.2.1 Коллаборативная фильтрация

При коллаборативной фильтрации оценка объекта вычисляется на основе предыдущих оценок и оценок похожих пользователей или объектов. Коллаборативная фильтрация содержит в себе две группы методов:

– Методы, основанные на работе с памятью – подразумевают вычисления с полноразмерной матрицей оценок. В ее основе лежит измерение сходства между пользователями или товарами. Соответственно, в ней выделяют два аналогичных по принципу работы подхода: фильтрацию по пользователям и фильтрацию по товарам.

– Методы, основанные на моделях – предполагают проведение предварительных операций над матрицей оценок, создание на ее основе описательной модели.

Преимуществом методов на основе моделей является то, что ресурсоемкие операции получения модели могут выполняться в отложенном режиме. В таком случае в режиме реального времени потребуется провести небольшое количество вычислений для обновления модели.

К методам этого класса относятся модели на основе скрытых факторов, влияющих на рейтинги пользователей, например, сингулярное разложение матрицы (SVD), неотрицательное матричное разложение (NMF), Стохастический градиентный спуск (SGD), метод чередующихся наименьших квадратов (ALS)

Также к методам на основе моделей могут относиться наивный байесовский классификатор, марковские цепи [7] (рисунок 2) и др.

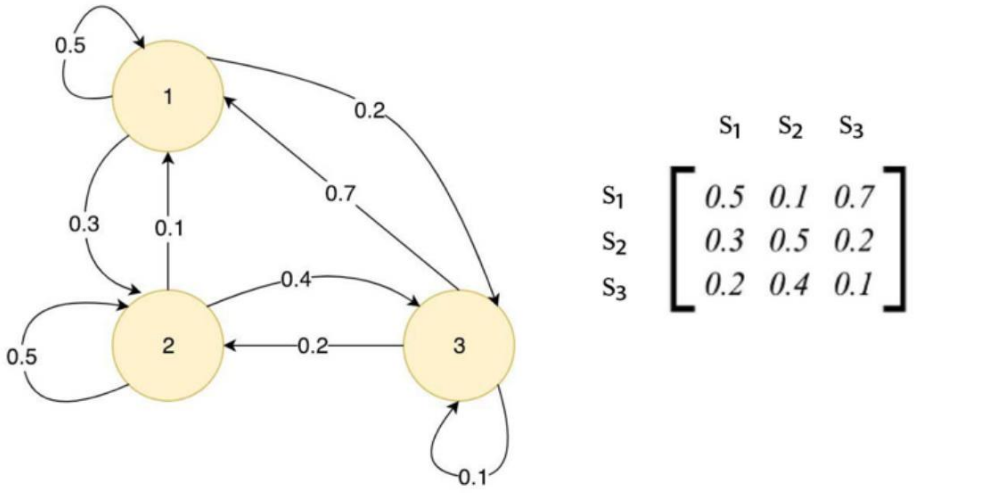

Рисунок 2 – Устройство марковской цепи

1.2.2 Фильтрация на основе содержимого

В фильтрации на основе содержимого большую роль играет информация об объектах, доступная системе. Например, это могут быть краткие описания к товарам, ключевые слова, оставляемые к записям в социальной сети и др. Системы, использующие данный метод, определяют предпочтительный объект исходя из того, какие категории объектов пользователь оценивает выше всего. Отличительной особенностью такого подхода является то, что рекомендации для пользователя не зависят от поведения других посетителей, и при работе с конкретным пользователем достаточно обработать лишь его данные.

Простой способ сопоставления пользователей и элементов – это сравнение элементов по ключевым словам. Например, для рекомендаций по вакансиям можно сопоставить описание работы с резюме соискателей..

Распространенным инструментом с данном типе систем является принцип TF-IDF [8] (TF — частота слова в описании, IDF — мера «редкости» слова в остальных описаниях) показывающий вес слова в описании, с помощью которого можно определить ключевые слова (рисунок 3).

Рисунок 3 – Формула расчета TF-IDF

1.2.3 Системы, основанные на знаниях

Системы, основанные на знаниях используют явные запросы пользователя, поскольку они предназначены для рекомендации товаров, которые покупаются нечасто (например, недвижимость, транспорт и т. п.), или для рекомендации товаров новым пользователям. В таких ситуациях невозможно просматривать историю покупок пользователя либо отталкиваться от выбора других клиентов. Для определения подходящего товара пользователь может либо указать критерии для интересующего его товара (подбор по ограничениям), либо выбрать наиболее близкий к его требованиям товар, и система подберет похожие (подбор по образцу) (рисунок 4).

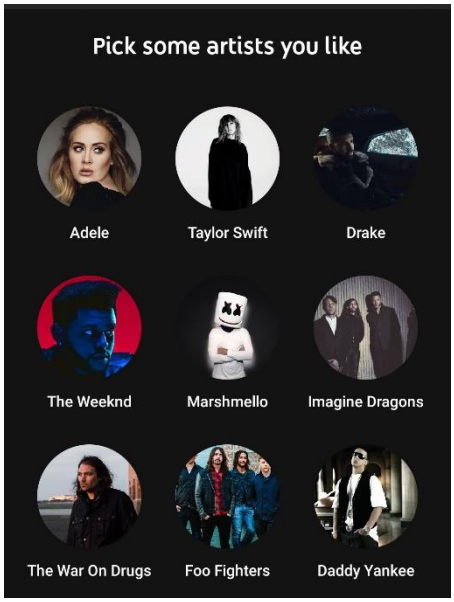

Рисунок 4 – Пример подбора по образцу в Spotify

1.2.4 Гибридные рекомендательные системы

Рекомендации, выданные гибридной системой, представляют собой комбинацию результатов систем прочих классов. Существует три основных типа систем: монолитные, комбинированные и смешанные.

Монолитные системы предполагают реализацию различных алгоритмов в одном модуле (рисунок 5).

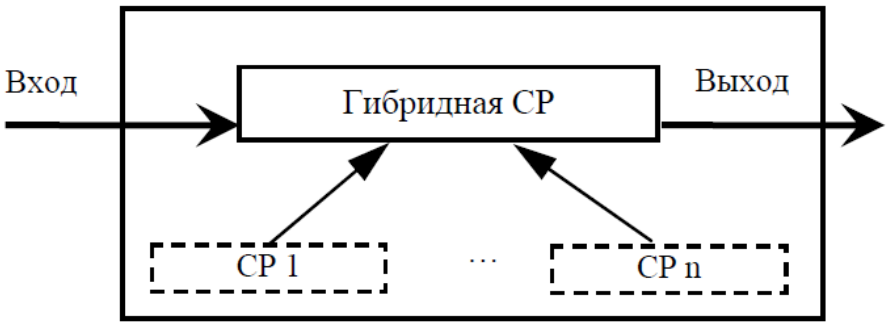

Рисунок 5 – монолитная организация системы

Комбинированные системы осуществляют обособленную работу алгоритмов, комбинируя в конце результаты или выдавая при определенном условии результат одной из систем (параллельная организация) (рисунок 6), либо передавая результаты первого алгоритма частично или полностью на вход второму и т. д. (последовательная организация) (рисунок 7).

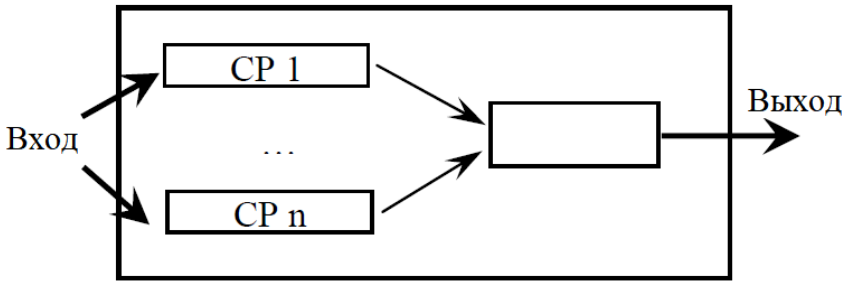

Рисунок 6 – Параллельная организация системы

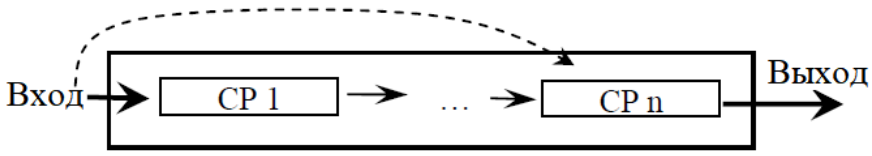

Рисунок 7 – Последовательная организация системы

Смешанные системы, в отличие от предыдущих, имеют несколько выходов и не комбинируют результаты, а показывают их вместе. Их особенность заключается в том, что рекомендации показываются в едином пространстве (странице браузера, окне программы и пр.).

2 Коллаборативная фильтрация, основанная на моделях

Методы коллаборативной фильтрации на основе моделей являются следующей ступенью в построении рекомендательных систем после фильтрации на основе сходства. Недостатком метода коллаборативной фильтрации, работающей с памятью, является относительно большое время выполнения операций над матрицей оценок, являющейся как правило разреженной и имеющей большой размер. Вследствие этого для снижения вычислительной нагрузки практикуется использование не прямых вычислений над оценками, а их предварительное приведение к модели. К подобным методам можно отнести методы понижения размерности векторов.

Задача понижения размерности сводится к замене матрицы оценок A размеров $u \times i$ на две матрицы: U размеров $u \times k$ и V размеров $i \times k$ [6]. Строки матрицы U содержат степени приверженности пользователя к k факторам. А строки матрицы V содержат степени соответствия товаров k факторам. В таком случае в памяти достаточно хранить не $u \times i$ значений, а $(u+i) \cdot k$. При этом, чтобы восстановить приблизительную оценку пользователя u продукту i, достаточно вычислить скалярное произведение строки пользователя из U и строки товара из V.

Возможны случаи, когда вычисляемые факторы могут иметь объяснение (например, что фактор отражает степень соответствия фильма жанру «нуар»). Однако их интерпретация зачастую неочевидна и нецелесообразна. По этой причине факторы называются скрытыми, или латентными.

К методам понижения размерности можно отнести сингулярное разложение матрицы, или SVD-разложение, неотрицательное матричное разложение (NMF), а также вероятностный латентный семантический анализ (PLSA).

2.1 Сингулярное разложение матрицы

Сингулярное разложение предполагает разложение матрицы оценок на три матрицы: факторы пользователей U размеров $u \times k$, факторы товаров V размеров $i \times k$ и диагональная матрица Σ размеров $k \times k$.

Данный метод основан на теореме о приведении матрицы к диагональной форме, имеющей следующую формулировку [9]:

«Для любой вещественной квадратной матрицы A размера $n \times n$ существуют две вещественные ортогональные $n \times n$ матрицы U и V, такие, что $U^T AV$ представляет собой диагональную матрицу Σ. При этом можно выбрать U и V так, чтобы диагональные элементы Σ имели вид:

$$\mu_1 \geq \mu_2 \geq \ldots \geq \mu_r > \mu_{r+1} = \ldots = \mu_n = 0,$$

где r – ранг матрицы A.

В случае, если матрица A – невырожденная, то $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_n > 0$. Столбцы матриц U и V называются соответственно левыми и правыми сингулярными векторами, а значения диагонали матрицы Σ называются сингулярными числами»

Из этой теоремы следует, что для любой матрицы A можно подобрать такие U и V^T, что получится разложение $A = U\Sigma V^T$, где Σ – диагональная матрица. В случае, когда рассматривается матрица A размера $u \times i$, получаемые матрицы U, Σ и V^T имеют размер $u \times u, u \times i$ и $i \times i$ соответственно (рисунок 8).

При разложении матрицы в системах рекомендаций используется так называемое экономное представление разложения. Оно выделено оранжевой заливкой на рисунке 8. В нем количество сингулярных значений уменьшается до k. Тогда, соответственно, размер матрицы Σ урезается до $k \times k$, а длина строк матриц U и V также сокращается до k. Так как элементы диагонали Σ

сортированы в порядке убывания, то потеря точности будет минимально возможной.

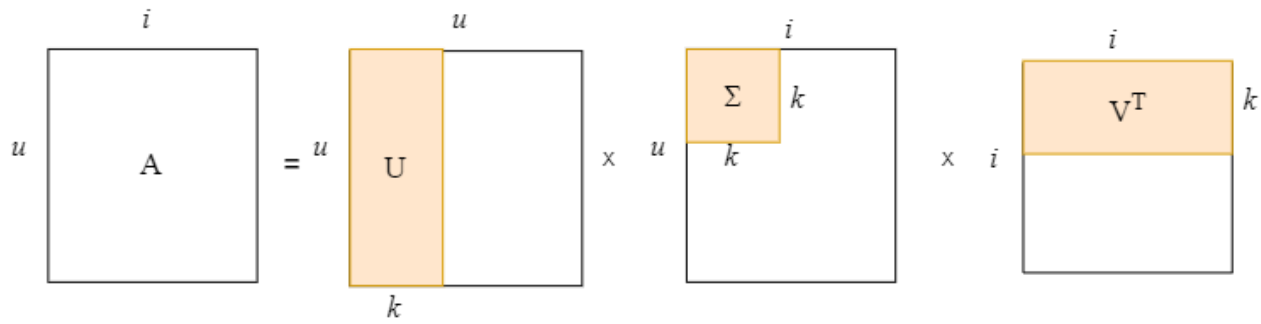

Рисунок 8 – Экономное и полное представления разложения

Количество факторов определяется в зависимости от задачи. Например, в книге [10] автор советует оставлять столько факторов, чтобы их сумма образовывала 90% от суммы всех имевшихся факторов. Малое количество факторов может дать недостаточную точность, в то время как большое их число может привести к переобучению модели.

Чтобы привести сингулярное разложение $A = U\Sigma V^T$ к общему виду $A = UV^T$, его можно представить как $A = (U\Sigma_{sqrt}) \times (\Sigma_{sqrt} V^T)$, где Σ_{sqrt} – диагональная матрица из квадратных корней сингулярных значений.

2.2 Метод стохастического градиентного спуска

Проблема метода сингулярного разложения заключается в том, что он позволяет прогнозировать оценки лишь для плотных матриц [11]. В случае работы с матрицей оценок с большим количеством пропусков этот метод может быть применен, если пропущенные значения будут заполнены средними оценками пользователя или продукта. Однако в таком случае предсказываемые оценки также будут близки к средним.

Более приемлемый метод для расчетов с большим количеством пропусков основан на подборе векторов факторов путем стохастического

градиентного спуска. Он был предложен Саймоном Фанком и имеет название Funk SVD. Хотя в его основе лежит не разложение матрицы, рейтинг в общей сути так же может быть выражен как произведение векторов факторов:

$$\hat{r}_{u,i} = p_u \cdot q_i^T,$$

где p_u и q_i – векторы факторов пользователя и товара соответственно.

Однако на практике произведения векторов p_u и q_i недостаточно. Рассматривая оценки пользователей, стоит также принимать во внимание факт, что каждый пользователь имеет свое представление о хорошей и плохой оценке, и от этого его оценки бывают завышены или занижены. Аналогично, каждый товар может иметь завышенные или заниженные оценки. Для того, чтобы компенсировать это смещение, при прогнозирования рейтинга используются базовые предикторы.

Каждый пользователь и каждый товар имеет свой предиктор b_u и b_i, характеризующий «предвзятость» оценок пользователя или товара. В простом случае они могут содержать центральные моменты первого порядка рейтингов пользователя или товара, модифицированные наличием смещения λ_1, λ_2 для возможности уменьшения предикторов [12]:

$$b_u = \frac{\sum_{i \in R(u)} \left(r_{u,i} - \mu \right)}{\lambda_1 + \left| R(u) \right|},$$

где $R(u)$ – множество товаров, оцененных пользователем u;

μ – среднее арифметическое всех имеющихся рейтингов.

$$b_i = \frac{\sum_{u \in R(i)} \left(r_{u,i} - \mu \right)}{\lambda_2 + \left| R(i) \right|},$$

где $R(i)$ – множество пользователей, оценивших товар i.

Соответственно, вычисление рейтинга с учетом предикторов выполняется по следующей формуле:

$$\hat{r}_{u,i} = \mu + b_u + b_i + p_u \cdot q_i^T . \quad (1)$$

Однако зачастую базовые предикторы так же, как векторы факторов, подбираются путем минимизации функции ошибки с регуляризацией:

$$\begin{aligned} L(b_u, b_i, p_u, q_i) &= \sum_{u,i}\left[\left(r_{u.i} - \mu - b_u - b_i - p_u \cdot q_i^T\right)^2 + \lambda\left(b_u^2 + b_i^2 + \left\|p_u\right\|^2 + \left\|q_i^T\right\|^2\right)\right] = \\ &= \sum_{u,i}\left[\left(r_{u.i} - \hat{r}_{u,i}\right)^2 + \lambda\left(b_u^2 + b_i^2 + \left\|p_u\right\|^2 + \left\|q_i^T\right\|^2\right)\right], \end{aligned} \quad (2)$$

где λ – параметр регуляризации.

Регуляризация используется для наложения штрафа на функцию ошибок, что позволяет избежать больших значений параметров в связи с переобучением.

Минимизация функции L реализуется методом стохастического градиентного спуска, как показано на формуле (3):

$$\begin{aligned} e_{u.i} &= r_{u.i} - \mu - b_u - b_i - p_u \cdot q_i^T, \\ b_u &\leftarrow b_u + \gamma \cdot \left(e_{u.i} - \lambda \cdot b_u\right), \\ b_i &\leftarrow b_i + \gamma \cdot \left(e_{u.i} - \lambda \cdot b_i\right), \\ p_u &\leftarrow p_u + \gamma \cdot \left(e_{u.i} \cdot q_i - \lambda \cdot p_u\right), \\ q_i &\leftarrow q_i + \gamma \cdot \left(e_{u.i} \cdot p_u - \lambda \cdot q_i\right), \end{aligned} \quad (3)$$

где γ – параметр, характеризующий скорость градиентного спуска.

Один шаг спуска включает в себя проход по рейтингам обучающей выборки. Для каждого рейтинга $r_{u,i}$ выполняется изменение предикторов b_u,

b_i и векторов факторов p_u и q_i путем прибавления производной текущего слагаемого в формуле (2).

В отличие от обычного градиентного спуска, в стохастическом для модификации параметров не нужно суммировать ошибки по всем рейтингам обучающей выборки. Для изменения параметра достаточно вычислить производную по нему лишь на одной оценке.

Градиентный спуск продолжается до тех пор, пока значение ошибки не начнет изменяться слишком мало, т. е. не будет выполнено условие $\left|L^{(k+1)} - L^{(k)}\right| < \varepsilon$.

3 Программная реализация методов понижения размерности

Для дальнейшего рассмотрения методов SVD-разложения матрицы и Funk SVD были написаны две программы на языке программирования Python: svd.py и svd_simon_funk.py. Они принимают на вход данные из csv-файла, в котором каждая строка имеет формат «ID пользователя, ID товара, рейтинг», и прогнозируют недостающие рейтинги.

3.1 Работа алгоритма сингулярного разложения матрицы

Программа svd.py использует алгоритм сингулярного разложения матрицы. В ней реализована работа с матрицами двух видов:

– матрица, составленная из файла типа csv;

– матрица, составленная из случайных значений.

SVD-разложение случайного набора данных было добавлено, чтобы проверить работу метода для плотной матрицы оценок.

В качестве csv-файла был выбран набор данных MovieLens, предоставляемый исследовательской лабораторией GroupLens Университета Миннесоты.

Работа программы проходит в следующих этапах:

– составление матрицы одним из двух способов;

– заполнение отсутствующих рейтингов пользователя его средней оценкой;

– получение матриц U, Σ и V^T и вычисление матрицы приблизительных оценок $\hat{A} = U\Sigma V^T$;

– если матрица получена из набора MovieLens, составляется и выводится ранжированный список фильмов, рекомендованных пользователю.

Для каждого типа прогнозирование рейтинга было выполнено в отдельных процедурах: recommend_from_random и recommend_from_csv.

Процедура recommend_from_random вначале формирует матрицу формата NumPy из случайных значений. Затем так же случайным образом определяются элементы, которые будут удалены из матрицы и будут считаться отсутствующими оценками. Позиции этих элементов заполняются средними оценками.

Далее выполняется цикл, реализующий разложение по различному числу факторов. В его теле производится разделение данных случайным образом на обучающую и тестовую выборки. Затем с помощью встроенной в NumPy функции linalg.svd вычисляется разложение обучающей выборки и приближенная матрица predicted_rating.

Полученные результаты и исходная матрица иллюстрируются в виде графика (рисунок 12) и набора т. н. «тепловых карт» матрицы (рисунок 9).

Процедура recommend_from_csv обладает схожей структурой, за исключением того, что вначале матрица формируется из объекта типа DataFrame библиотеки Pandas. DataFrame является структурой данных, представляющей собой таблицу. Аналогично процедуре recommend_from_random, пропущенные значения таблицы заполняются средними оценками. Столь большой объем получившейся заполненной матрицы может вызвать проблемы при работе с функциями, поэтому необходимо накладывать ограничения на количество прочитанных из csv-файлов строк.

Полученная матрица разделяется на обучающую матрицу и тестовую, и в цикле для различного числа факторов вычисляется матрица приблизительных оценок, среднеквадратическое отклонение и графики оценок.

3.1.1 Результат работы сингулярного разложения матрицы

Для рассмотрения результатов работы программ были построены различные графики с использованием библиотеки matplotlib.

Например, как отмечалось выше, работа SVD-разложения осложнена для разреженных матриц. Действительно, на рисунке 9 представлена работа алгоритма в таком случае, и видно, что нулевые значения приближаются к нулю:

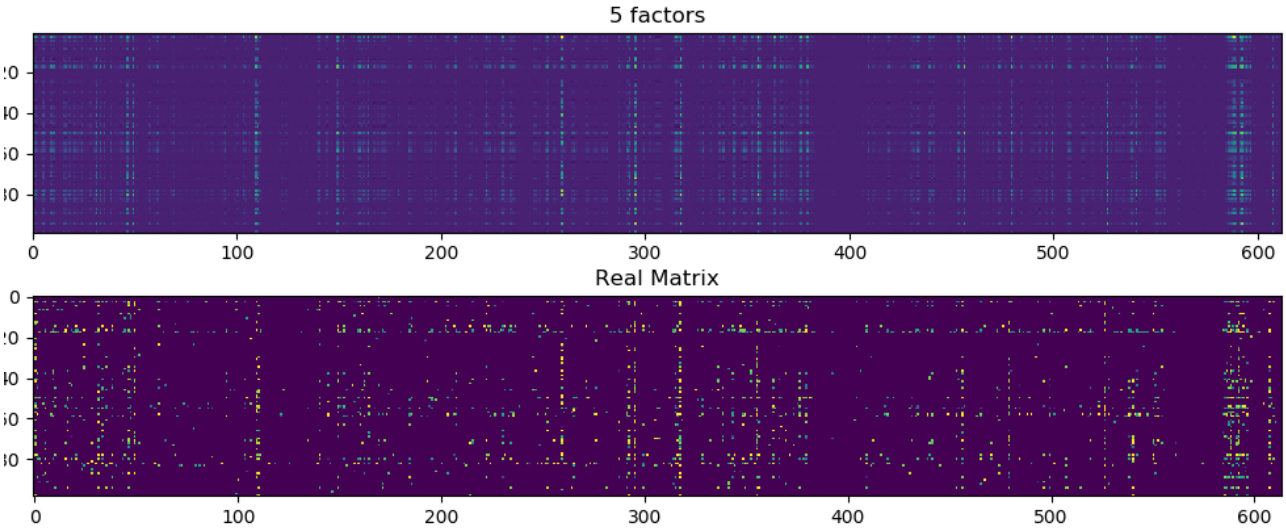

Рисунок 9 – SVD-разложение в случае разреженной матрицы

Если в матрице заполнить пропуски средними оценками пользователя, то недостающие элементы матрицы оценок будут также приближаться к среднему рейтингу, однако рядом с элементами, где оценки представлены, возможны приближения к этим оценкам (рисунок 10). Также на рисунке 10 хорошо видно, что при малом числе факторов переход между элементами плавный, а сами элементы близки к средней оценке.

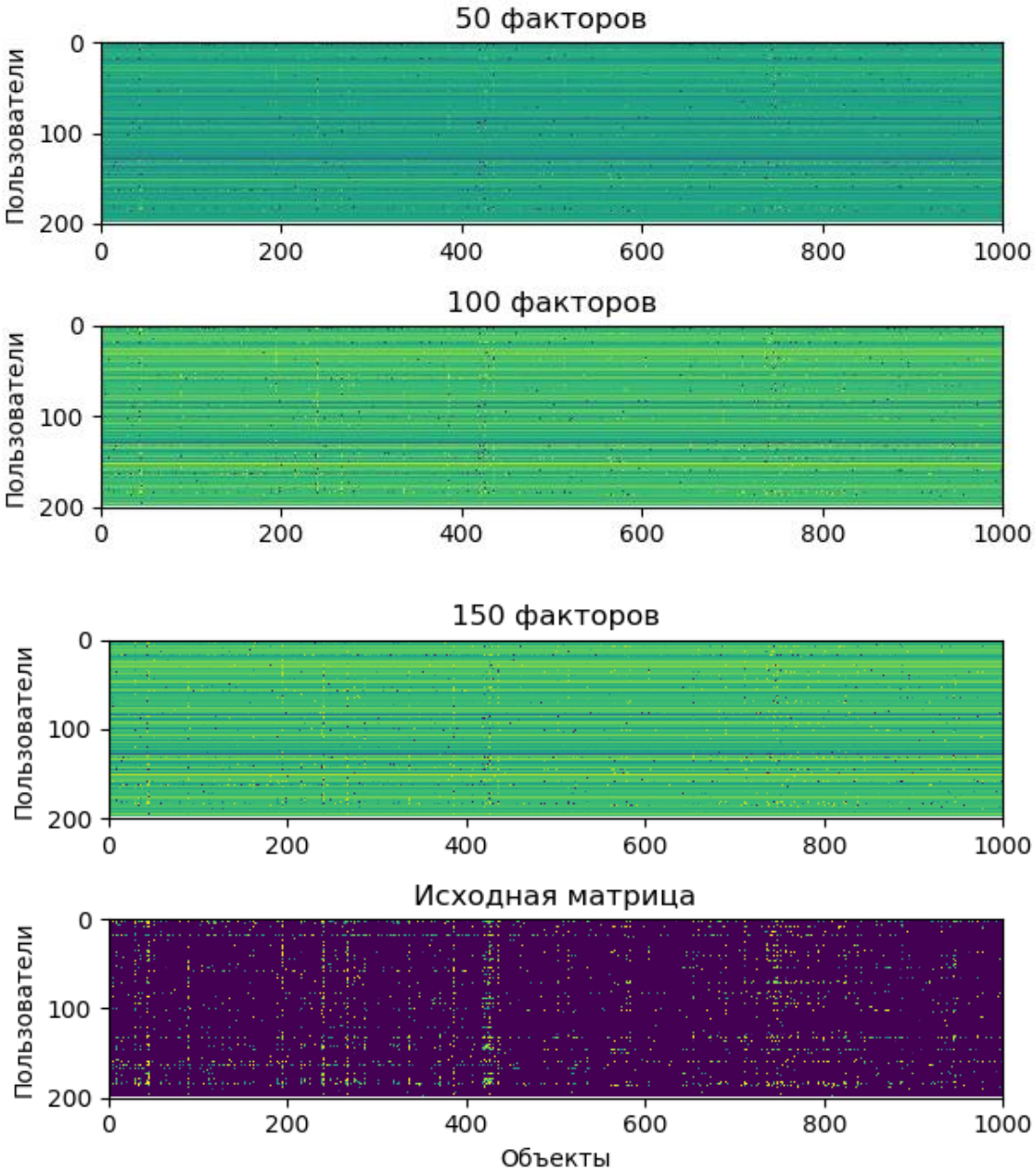

Рисунок 10 – SVD-разложение в случае заполнения разреженной матрицы средними значениями

Соответственно, если вывести ранжированный список рекомендуемых фильмов, можно увидеть, что их рейтинги близки к средним (рисунок 11).

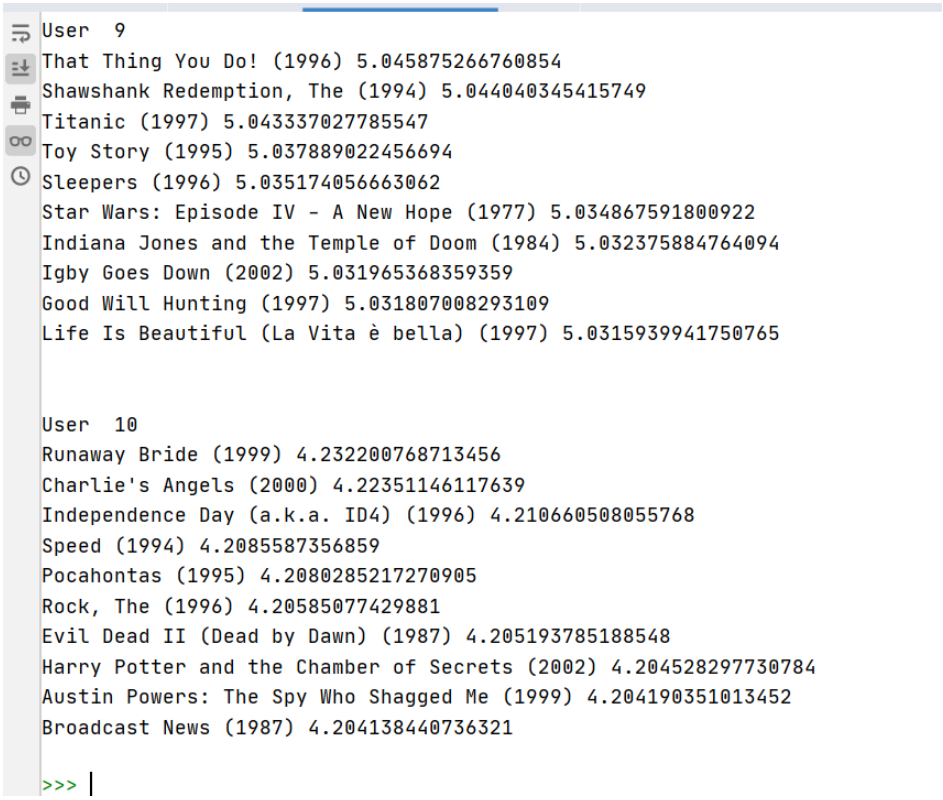

Рисунок 11 – Список рекомендуемых фильмов

Рассмотрим, с другой стороны, более плотную матрицу, полученную заполнением элементов случайными значениями от 1 до 5. Для нее алгоритм сингулярного разложения выполняет неплохое приближение, если заполнить ее элементы средними значениями. Хотя заметны колебания оценок, общее поведение схоже со входной матрицей. Это видно на рисунке 12, на котором показаны оценки пользователя 1 в порядке их возрастания в поданной на вход матрице с заполненными средними значениями. Однако при большом числе факторов модель склонна к переобучению. Так, при 100 факторах из 100 возможных, алгоритм в большей мере повторяет входную матрицу.

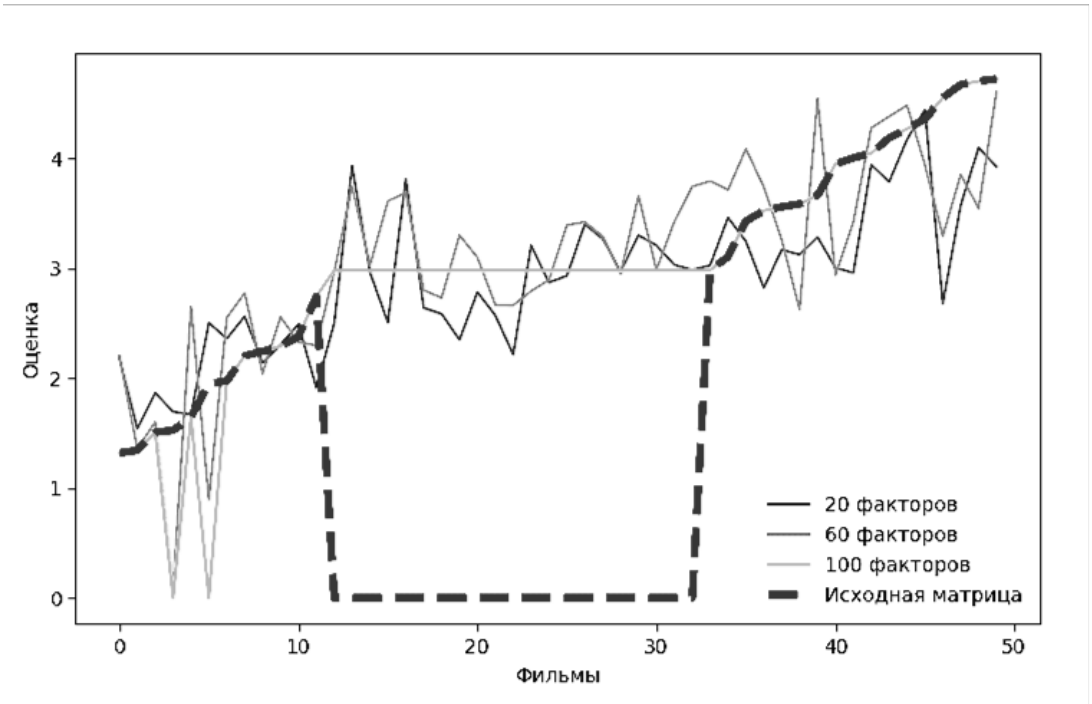

Рисунок 12 – График оценок пользователя 1

3.2 Работа алгоритма стохастического градиентного спуска

Вторая программа – svd_simon_funk.py – реализует поиск факторов методом стохастического градиентного спуска. В отличие от предыдущей программы, алгоритм Funk SVD не требует составления матрицы, что позволяет работать напрямую с таблицей DataFrame. В ней выполняются следующие блоки:

– создание таблицы DataFrame из csv-файла;

– инициализация констант и параметров;

– разделение данных на обучающую и тестовую таблицы;

– вычисление рейтингов для тестовой таблицы;

– расчет среднеквадратической ошибки и построение графиков реальных и предсказанных рейтингов для тестовой таблицы.

Программа начинает работу с получения таблицы DataFrame из набора рейтингов к фильмам, после чего объявляются параметры b_u, b_i как NumPy-массивы и p, q в виде NumPy-матриц.

Каждый элемент b_u и b_i – это базовый предиктор пользователя и фильма соответственно. На начальном этапе они содержат средние отклонения рейтингов соответствующего пользователя либо фильма от средней оценки всей таблицы, умноженные на 0,01.

Матрицы p и q имеют размер $u \times k$ и $i \times k$ соответственно и содержат векторы факторов пользователя и фильма. На начальном этапе они заполнены случайными значениями из нормального распределения от 0 до 0,1.

Затем производится разбиение индексов таблицы на обучающую и тестовую выборки в отношении 9:1, и в соответствии с индексами создается обучающая таблица и тестовая таблица.

На обучающей таблице выполняется функция стохастического градиентного спуска Gradient_descent, которая возвращает измененные параметры b_u, b_i, p и q, принимая вместе с ними константы $\lambda = 0,2$, $\gamma = 0,05$ и $\varepsilon = 0,01$.

В теле функции Gradient_descent выполняется цикл, в котором в k-й итерации вычисляется значение функции $L^{(k)}(b_u, b_i, p_u, q_i)$. Также в цикле производится поэтапное уменьшение скорости обучения γ на 60%, если разность $L^{(k+1)} - L^{(k)}$ становится меньше порога, установленного в переменной threshold. Порог threshold. в свою очередь, также уменьшается на 50%, чтобы отследить, когда в следующий раз необходимо уменьшить параметр γ. Начальное значение переменной threshold равно 5.

Программа вычисляет функцию $L^{(k)}(b_u, b_i, p_u, q_i)$ в отдельной процедуре по формуле (3), в процессе прохода по оценкам изменяя значения параметров согласно формуле (4).

Далее вычисляются рейтинги для тестовой выборки, после чего результаты оформляются в виде графика, как показано на рисунке 13.

В конце программа проходит по номерам пользователей и фильмов, и если оценка отсутствует в таблице, то вычисляется приближенная оценка.

Полученные рейтинги сортируются, и в отдельной процедуре выводится список фильмов с наивысшими оценками.

3.2.1 Результат работы стохастического градиентного спуска

Результаты работы программы были оформлены в виде графика. На рисунке 13 представлены элементы тестовой выборки. черной линией отмечены настоящие рейтинги тестовой выборки, серой – приблизительные рейтинги, полученные по формуле (2).

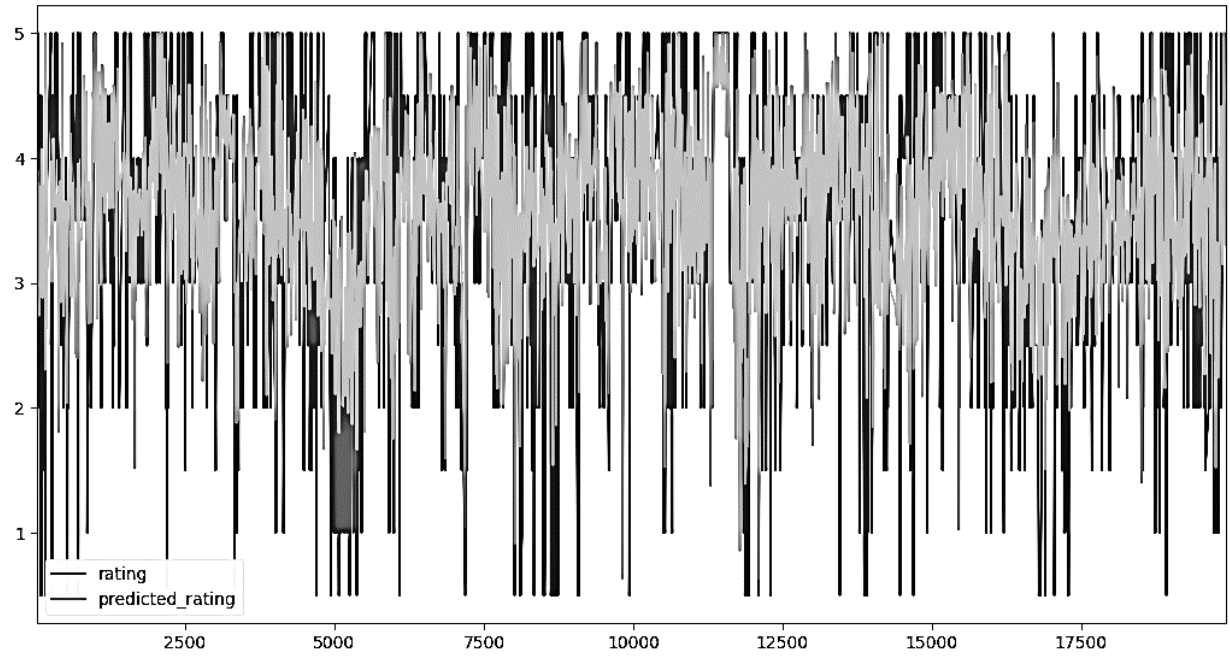

Рисунок 13 – График сравнения вычисленных рейтингов в тестовой выборке

Практически при всех запусках программы среднеквадратическая ошибка предсказания рейтингов колеблется от 0,94 до 0,98. Однако в процессе вывода списков фильмов с наивысшим рейтингом было замечено, что в них часто оказываются фильмы, для которых был подобран наивысший предиктор b_i. Подобная ситуация частично исправляется тем, что в цикле, вместе со

скоростью обучения γ, на 60% уменьшается параметр регуляризации λ, когда разность $L^{(k+1)} - L^{(k)}$ становится меньше порога threshold.

Вывод программы представлен на рисунке 14:

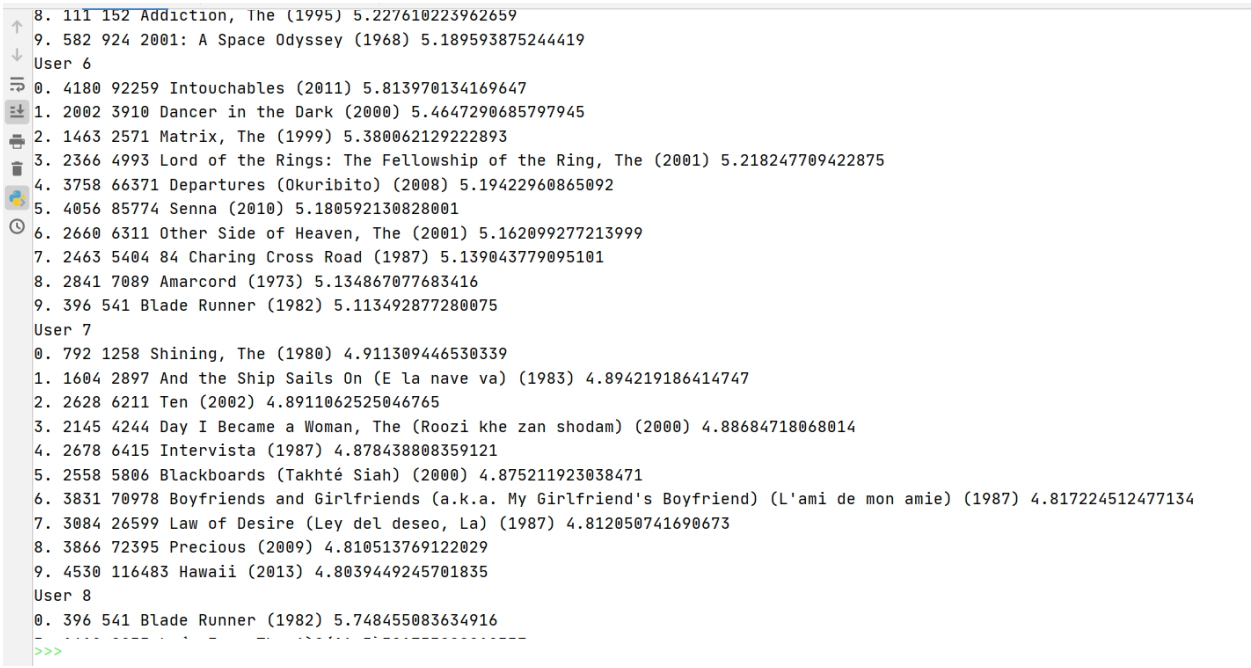

Рисунок 14 – Вывод фильмов с наивысшим рейтингом

ЗАКЛЮЧЕНИЕ

В данной курсовой работе были изложены теоретические сведения о рекомендательных системах. Описано историческое развитие систем рекомендаций, были описаны классы моделей, используемых в разработке рекомендательных систем.

Во втором разделе работы были изучены и описаны методы понижения размерности матрицы: сингулярное разложение и стохастический градиентный спуск. Были описаны теоретические основы для программной реализации данных алгоритмов.

В третьем разделе модели были реализованы на языке программирования Python. В программах составлены графики работы алгоритмов и выведен результат в виде ранжированного списка. Был выполнен анализ результатов работы программ.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Michael D. Ekstrand. Collaborative Filtering Recommender Systems. / Michael D. Ekstrand John T. Riedland Joseph A. Konstan. // Foundations and Trends in Human-Computer Interaction – 2010 – Vol. 4, No. 2. – с. 81-173.

2 Maxim Naumov. Dheevatsa Mudigere. DLRM: An advanced, open source deep learning recommendation model. 02.07.2019. URL: https://ai.facebook.com/blog/dlrm-an-advanced-open-source-deep-learning-recommendation-model/ (Дата обращения 16.06.2020)

3 Ivan Medvedev, Haotian Wu, Taylor Gordon. Powered by AI: Instagram's Explore recommender system. 25.11.2019. URL: https://ai.facebook.com/blog/powered-by-ai-instagrams-explore-recommender-system/ (Дата обращения 16.06.2020)

4 Антон Волнухин. Платформа «Атом» от Яндекса — интернет, адаптированный для каждого / Блог компании Яндекс / Хабр. URL: https://habr.com/ru/company/yandex/blog/195982/ (Дата обращения 16.06.2020).

5 Меньшикова Н. В. Обзор рекомендательных систем и возможностей учета контекста при формировании индивидуальных рекомендаций / Меньшикова Н. В., Портнов И. В., Николаев И. Е. // Academy. – 2016. – №6 (9). – с. 20-22.

6 Aggarwal C. C. Recommender Systems: The Textbook / C. C. Aggarwal. – NY: Springer International Publishing, 2016 – 498 с.

7 Jonathan Hui. Machine Learning — Recommender System. 10.02.2020. URL: https://medium.com/@jonathan_hui/machine-learning-recommender-system-e3237b9df14a (дата обращения 10.06.2020).

8 Анатомия рекомендательных систем. Часть первая / Блог компании ГК ЛАНИТ / Хабр. – URL: https://habr.com/ru/company/lanit/blog/420499/ (дата обращения 10.06.2020).

9 Форсайт Дж. Численное решение систем линейных алгебраических уравнений / Дж. Форсайт, К. Молер. – М.: Издательство «Мир», 1969. – 167 с.

10 Falk K. Practical Recommender Systems / K. Falk. – NY: Manning Publications, 2019 – 432 с.

11 Nicolas Hug. Understanding matrix factorization for recommendation (part 3) – SVD for recommendation. 16.06.17. URL: http://nicolas-hug.com/blog/matrix_facto_3 (дата обращения 16.05.2020).

12 Ricci F. Recommender Systems Handbook / F. Ricci, L. Rokach, B. Shapira [et al.]. – NY: Springer, 2011 – 842 с.

ПРИЛОЖЕНИЕ

Содержимое файла svd.py

```
from time import time

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from os import path
from sklearn.metrics import mean_squared_error
from math import sqrt


def train_test_split(matrix, test_percent, isnull_table, replace_test_with):
    """ Разделение данных на обучающие и тестовые """

    test_matrix = np.zeros_like(matrix)
    train_matrix = matrix.copy()
    np.random.seed(1234)

    for user_num in range(train_matrix.shape[0]):
        count_nz = np.count_nonzero(~isnull_table[user_num])  # isnull_table - это
bool-массив, опред. наличие рейтинга
        nz_choice = np.nonzero(~isnull_table[user_num])[0]  # массив, содержащий
индексы не null элементов

        # выбор случайных значений из массива индексов
        test_choice = np.random.choice(nz_choice, size=int(test_percent * count_nz),
replace=False)
        test_matrix[user_num, test_choice] = train_matrix[user_num, test_choice]
        train_matrix[user_num, test_choice] = replace_test_with[user_num]

    return train_matrix, test_matrix


def svd(matrix, k):
    """ SVD-разложение по k факторам """

    u, s, vt = np.linalg.svd(matrix, full_matrices=False)
    sigma = np.diag(s)
    return np.dot(np.dot(u[:, :k], sigma[:k, :k]), vt[:k, :])


def rmse(predicted, test):
    """ Среднеквадратическое отклонение """

    test_nz_choice = np.nonzero(test.flatten())
    mse = mean_squared_error(predicted.flatten()[test_nz_choice],
test.flatten()[test_nz_choice])
    print(predicted.flatten()[test_nz_choice[:50]],
test.flatten()[test_nz_choice[:50]])
    return sqrt(mse)




def get_table_from_csv():
    """ Составление таблицы (dataframe) рейтингов из csv-файла,
```

```
        а также словаря средних оценок пользователя """

    dataset_path = path.join(r'D:\Универ\!!Курсовая рекоменд системы\program\ml-
latest', 'ratings.csv')
    ratings = pd.read_csv(dataset_path, delimiter=',', header=0, usecols=['userId',
'movieId', 'rating'])
    ratings = ratings[ratings['userId'] < 200]

    mean_ratings = ratings[['userId', 'rating']].groupby('userId').mean()
    mean_dict = mean_ratings.to_dict()['rating']  # словарь вида { userId :
mean_rating }

    rating_table = ratings.pivot(index='userId', columns='movieId', values='rating')
    return rating_table, mean_dict


def print_recommendations(predicted_rating, rating_table, top_size):
    """ Вывод топа рекомендуемых элементов """

    isnull_table = rating_table.isnull().values
    without_rated = predicted_rating
    np.putmask(without_rated, np.invert(isnull_table), 0)  # рейтинги просмотренных
фильмов ставятся в 0

    # Индексы рекомендуемых фильмов
    top = without_rated.argsort(axis=1)
    top = top[:, ::-1]
    top = top[:, :top_size]

    def prn_top(top_array):
        movies_path = path.join(r'D:\Универ\!!Курсовая рекоменд системы\program\ml-
latest', 'movies.csv')
        movie_df = pd.read_csv(movies_path, delimiter=',', header=0,
usecols=['movieId', 'title'])

        # Проход по матрице топов
        for u in range(top_array.shape[0]):
            user_id = rating_table.index[u]
            print('\n\nUser ', user_id)
            top_mov_idxs = top_array[u]

            # Вывод названий фильмов
            for mov_idx in top_mov_idxs:
                movie_id = rating_table.columns[mov_idx]  # id фильма по индексу
                movie_df_row = movie_df[movie_df['movieId'] == movie_id]
                rating = rating_table.loc[user_id, movie_id]
                print((movie_df_row['title'].tolist())[0], predicted_rating[u,
mov_idx])

    prn_top(top)


def recommend_from_csv():
    """ Основная программа """

    rating_table, mean_dict = get_table_from_csv()
    isnan_table = np.isnan(rating_table.values)

    table_np = rating_table.T.fillna(
        value=mean_dict).to_numpy().T  # Заменяем NaN-значения средними и создаем
питру-матрицу рейтингов
```

```
    # Составляются разложения с 1..[кол-во пользователей] факторами
    fig = plt.figure()  # создается рисунок
    fig2 = plt.figure()

    predicted_rating = np.empty_like(table_np)
    factor_num_list = list(range(50, table_np.shape[0]+1, 50))
    arg_sort = table_np.argsort()

    for i, factor_num in enumerate(factor_num_list):
        train_table, test_table = train_test_split(table_np, test_percent=0.1,

isnull_table=rating_table.isnull().values,

replace_test_with=list(mean_dict.values()))
        predicted_rating = svd(train_table, factor_num)
        print('\nRMSE when {} factors = '.format(factor_num), rmse(predicted_rating,
test_table))

        # Добавляется изобр. матрицы
        ax = fig.add_subplot(len(factor_num_list) + 1, 1, i + 1)
        ax.imshow(predicted_rating)
        ax.set_title('%i факторов' % factor_num)
        ax.set_ylabel('Пользователи')

        ax = fig2.add_subplot()
        ax.plot(predicted_rating[4, arg_sort[0]], label='%i факторов' % factor_num)
        ax.set_ylabel('Пользователи')

    # Замена неоцененных объектов на 0
    source_table = table_np # np.sort(table_np, axis=1)
    np.putmask( source_table, isnan_table, 0 )

    ax = fig.add_subplot(len(factor_num_list) + 1, 1, len(factor_num_list) + 1)
    ax.imshow(source_table)
    ax.set_title('Исходная матрица')
    ax.set_xlabel('Объекты')
    ax.set_ylabel('Пользователи')

    ax = fig2.add_subplot()
    ax.scatter(np.arange(source_table.shape[1]), source_table[4, arg_sort[0]],
label='Исходные оценки')
    ax.set_xlabel('Объекты')
    ax.set_ylabel('Оценка')
    ax.legend()


    plt.show()

    print_recommendations(predicted_rating, rating_table, 10)

def recommend_from_random():
    """ Прогнозирование рейтингов на матрице случайных значений """

    def get_ratings_random(user_num, item_num):
        """ Генерация матрицы оценок из случайных значений """

        np.random.seed(1111)
        table_rand = np.random.rand(user_num, item_num) * 4 + 1

        # выбор значений, которые будут "пропущены"
```

```
        for usr_row in table_rand:
            tozero_idx = np.random.choice(len(usr_row),
np.random.randint(int(item_num / 5), int(item_num / 2)),
                                          replace=False)
            np.put(usr_row, tozero_idx, np.nan)

        isnan_table = np.isnan(table_rand)
        mean = np.empty(table_rand.shape[0])

        # замена NaN-значений средними
        for ind, usr_row in enumerate(table_rand):
            mean[ind] = usr_row[~isnan_table[ind]].mean()
            usr_row[isnan_table[ind]] = mean[ind]

        return table_rand, isnan_table, mean

    table_np, isnan_table, mean = get_ratings_random(100, 500)

    # Составляются разложения с 1..[кол-во пользователей] факторами
    fig = plt.figure()  # создается рисунок
    fig2 = plt.figure()

    factor_num_list = list(range(20, table_np.shape[0]+1, 40))
    print(table_np.shape)
    arg_sort = table_np[0, :50].argsort() # сортировка по величине рейтингов

    for i, factor_num in enumerate(factor_num_list):
        train_table, test_table = train_test_split(table_np, test_percent=0.1,
                                                   isnull_table=isnan_table,

replace_test_with=np.zeros_like(mean))
        predicted_rating = svd(train_table, factor_num)
        print('\nRMSE when {} factors = '.format(factor_num), rmse(predicted_rating,
test_table))

        # Добавляется изобр. матрицы
        ax = fig.add_subplot(len(factor_num_list)+1, 1, i+1)
        ax.imshow(predicted_rating)
        ax.set_title('%i факторов' % factor_num)
        ax.set_ylabel('Пользователи')

        ax = fig2.add_subplot()
        ax.plot(predicted_rating[0, arg_sort], label='%i факторов' % factor_num)
        ax.set_ylabel('Пользователи')

    # Замена неоцененных объектов на 0 (для показа на графике)
    source_table = table_np # np.sort(table_np, axis=1)
    np.putmask( source_table, isnan_table, 0 )

    ax = fig.add_subplot(len(factor_num_list)+1, 1, len(factor_num_list)+1)
    ax.imshow(source_table)
    ax.set_title('Исходная матрица')
    ax.set_xlabel('Объекты')
    ax.set_ylabel('Пользователи')

    ax = fig2.add_subplot()
    ax.plot(source_table[0, arg_sort],
            linestyle='--', linewidth=4, label='Исходная матрица')
    ax.legend()
    ax.set_xlabel('Объекты')
    ax.set_ylabel('Оценка')
```

```
    plt.show()


""" Две функции прогнозирования рейтинга.
    Предлагается одну из них комментировать """
startTimer = time()

# recommend_from_random()
recommend_from_csv()

stopTimer = time()
print('\nTime: ', stopTimer - startTimer, ' seconds')
```

Содержимое файла svd_simon_funk.py

```
from time import time
from os import path

import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
from math import sqrt
from matplotlib import pyplot as plt


def Gradient_descent(ratings, bu, bi, p, q, _lambda=2.0, _gamma=0.1, _eps=0.01):

    def Error(bu, bi, p, q, _lambda=0.001, _gamma=0.01):
        """ Функция ошибки, которую надо мимнимизировать """

        # В результате цикла будет посчитана сумма ошибок и параметры спустятся на 1
шаг
        penalted_error_sum = 0
        for usr_id, mov_id, rat in zip(ratings['userId'].values,
ratings['movieId'].values, ratings['rating'].values):
            user_ind = usr_indexes[usr_id]
            movie_ind = mov_indexes[mov_id]

            fit_rating = mean_total + bu[user_ind] + bi[movie_ind] +
np.dot(q[movie_ind], p[user_ind])
            err = rat - fit_rating
            penalted_error_sum += err**2 + \
            _lambda * (bu[user_ind]**2 + bi[movie_ind]**2 + np.sum(p[user_ind]**2) +
np.sum(q[movie_ind]**2))

            bu[user_ind] += _gamma * (err - _lambda * bu[user_ind])
            bi[movie_ind] += _gamma * (err - _lambda * bi[movie_ind])
            p[user_ind] += _gamma * (err * q[movie_ind] - _lambda * p[user_ind])
            q[movie_ind] += _gamma * (err * p[user_ind] - _lambda * q[movie_ind])

        return penalted_error_sum

    # Цикл градиентного спуска

    end_condition = False
    iternum = 1
    threshold = 5
```

```
    error = 0
    while not end_condition:
        old_error = error
        error = Error(bu, bi, p, q, _lambda=_lambda, _gamma=_gamma)

        if abs(old_error-error) < threshold:
            _gamma *= 0.4
            _lambda *= 0.4
            threshold *= 0.5
            print('_GAMMA={}, threshold={}'.format(_gamma, threshold))

        print('{}. Diff={}'.format(iternum, abs(error - old_error)))
        end_condition = iternum > 500 or abs(error - old_error) < _eps
        iternum +=1

    return bu, bi, p, q


startTimer = time()

# Открытие набора данных и подготовка индексов
dataset_path = path.join(r'D:\Универ\!!Курсовая рекоменд системы\program\ml-latest',
'ratings.csv')
ratings_df = pd.read_csv(dataset_path, delimiter=',', header=0, usecols=['userId',
'movieId', 'rating'])

# Соответствие id объектов и индексов в массивах
usr_ids = ratings_df['userId'].unique()
usr_indexes = {id: index for index, id in enumerate(usr_ids)}
mov_ids = np.sort(ratings_df['movieId'].unique())
mov_indexes = {id: index for index, id in enumerate(mov_ids)}

# Определение констант
FACTOR_NUM = 40
_LAMBDA = 0.2
_GAMMA = 0.05
EPS = 0.01

# Инициализация предикторов

q = np.random.normal(0.0, 0.1, (mov_ids.size, FACTOR_NUM))  # вектор факторов
пользователя
p = np.random.normal(0.0, 0.1, (usr_ids.size, FACTOR_NUM))  # вектор факторов фильмов

mean_total = ratings_df['rating'].mean()
ratings_df.insert(ratings_df.shape[1], 'rating_deviation',
ratings_df['rating'].values - mean_total)

bu = ratings_df.groupby('userId').mean()['rating_deviation'].values * 0.01 #
предиктор пользователя
bi = ratings_df.groupby('movieId').mean()['rating_deviation'].values * 0.001 #
предиктор фильма

kf = KFold(n_splits=10, shuffle=True, random_state=None)
cv_iterator = kf.split(ratings_df)

train_ind, test_ind = next(cv_iterator)
train_df = ratings_df.iloc[train_ind]
test_df = ratings_df.iloc[test_ind]

bu, bi, p, q = Gradient_descent(train_df, bu, bi, p, q, _lambda=_LAMBDA,
```

```
_gamma=_GAMMA, _eps=EPS)

sum_err = 0
predicted_rating_arr = np.empty(test_df.shape[0])
ind_arr = 0
for usr_id, mov_id, rat in zip(test_df['userId'].values, test_df['movieId'].values,
test_df['rating'].values):
    user_ind = usr_indexes[usr_id]
    movie_ind = mov_indexes[mov_id]

    predicted_rating = mean_total + bu[user_ind] + bi[movie_ind] +
np.dot(q[movie_ind], p[user_ind])

    predicted_rating_arr[ind_arr] = predicted_rating
    if predicted_rating > 5.0:
        predicted_rating_arr[ind_arr] = 5.0
    if predicted_rating < 0:
        predicted_rating_arr[ind_arr] = 0

    ind_arr += 1
    err = abs(rat - predicted_rating)
    sum_err += err**2
    print(usr_id, mov_id, rat, predicted_rating, err)

mean_squared_error = sum_err / test_df.shape[0]
print('RMSE={}'.format(sqrt(mean_squared_error)))


# построение графика
test_df.insert(test_df.shape[1], 'predicted_rating', predicted_rating_arr) #
добавление предсказанных рейтингов
test_df.plot(kind='line', use_index=True, y=['rating', 'predicted_rating'])
plt.show()


def print_recommendations(usr_id, usr_list, top_size):
    """ Вывод топа рекомендуемых элементов """

    top = usr_list[:top_size]
    mov_list = [tpl[0] for tpl in top]

    movies_path = path.join(r'D:\Универ\!!Курсовая рекоменд системы\program\ml-
latest', 'movies.csv')
    movie_df = pd.read_csv(movies_path, delimiter=',', header=0, usecols=['movieId',
'title'])

    # Вывод названий фильмов
    print('User', usr_id)
    for i, mov_tpl in enumerate(top):
        movie_df_row = movie_df[movie_df['movieId'] == mov_tpl[0]]
        predicted_rating = mov_tpl[1]
        print('{}. {} {} {} {}'.format(i, mov_indexes[mov_tpl[0]], mov_tpl[0],
movie_df_row['title'].tolist()[0], mov_tpl[1]))


# Проход по отсутсвующим в таблице парам пользователь - фильм
for usr_ind in range(usr_ids.size):
    usr_id = usr_ids[usr_ind]
    usr_list = []

    for mov_ind in range(mov_ids.size):
```

```
        mov_id = mov_ids[mov_ind]
        if not ((ratings_df['userId']==usr_id) &
(ratings_df['movieId']==mov_id)).any():
            predicted_rating = mean_total + bu[usr_ind] + bi[mov_ind] +
np.dot(q[mov_ind], p[usr_ind])
            usr_list.append((mov_id, predicted_rating))

    usr_list.sort(key=lambda x: x[1], reverse = True)
    print_recommendations(usr_id, usr_list, 10)

stopTimer = time()
print('\nTime: ', stopTimer - startTimer, ' seconds')
```