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, ЛЕЖАЩИХ В ОСНОВЕ РЕКОМЕНДАТЕЛЬНЫХ СИСТЕМ

Работу выполнил _____ А. Р. Чениб
(подпись)

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

Направленность Технология программирования

Научный руководитель
канд. физ.-мат. наук, доц. _____ В. Н. Кармазин
(подпись)

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

Краснодар
2019

РЕФЕРАТ

Курсовая работа содержит 33 страницы, 7 рисунков, 11 источников.

РЕКОМЕНДАТЕЛЬНЫЕ СИСТЕМЫ, РЕКОМЕНДАЦИИ, ЭЛЕКТРОННЫЙ РЕСУРС, КОЛЛАБОРАТИВНАЯ ФИЛЬТРАЦИЯ, ПОЛЬЗОВАТЕЛЬ, МЕРА БЛИЗОСТИ, МАТРИЦА ОЦЕНОК, ТРАНЗИТИВНЫЕ АССОЦИАТИВНЫЕ СЕТИ, ГРАФ

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:

– обработка данных для получения матрицы оценок пользователей;

– подготовка вспомогательных данных для предсказания оценок пользователей;

– реализация алгоритма коллаборативной фильтрации, основанной на работе с памятью;

– реализация алгоритма создания транзитивной ассоциативной сети пользователей и фильмов;

– вывод полученных данных в виде ранжированного списка рекомендуемых фильмов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

За последние двадцать лет рекомендательные системы успели стать неотъемлемой частью электронной торговли. С тех пор, как появилась возможность проводить автоматизированный сбор и обработку информации, в бизнесе сформировалась новая потребность в анализе данных. Это позволило бы как классифицировать потребности клиентов, так и более точно и оперативно адаптировать под них перечень продуктов, благодаря чему фирма может оптимизировать свои денежные затраты.

Рекомендательные системы являются одним из приложений анализа данных. Они используются для выявления пользовательский предпочтений и составления с их учетом персонализированного каталога товаров.

В настоящее время рекомендательные системы широко используются в интернет-бизнесе, при этом область их применения не ограничена интернет-магазинами. Любая электронная площадка, предоставляющая пользователю какой бы то ни было контент, приходит к задаче предоставить посетителю информацию, которая его больше заинтересует. Так, помимо торговых площадок в интернете, подобным Amazon и Alibaba, рекомендательные системы успешно применяются в медиахостингах (YouTube, Netflix, last.fm, Яндекс.Музыка), сайтах-агрегаторах (Flipboard, Яндекс.Маркет) и социальных сетях (Вконтакте, Facebook, Instagram).

При столь большом разнообразии вариантов применения рекомендательных систем перед разработчиками встает необходимость учитывать специфику конкретной задачи. Соответственно, существует множество типов систем рекомендаций, подходящих для той или иной ситуации. В данной курсовой работе будут рассмотрены основные концепции, используемые в разработке рекомендательных систем, а так же будет создана программа на языке программирования Python, реализующая рекомендацию фильмов пользователям по известному набору данных с использованием двух различных алгоритмов.

1 Сведения о предметной области

Рекомендательные системы являются распространенным применением инструментов анализа данных. Относясь к системам фильтрации информации, они призваны решать проблему избытка контента, с которой может столкнуться пользователь сервиса.

Чтобы он не потерял интерес к ресурсу, можно воспользоваться неперсонализированными рекомендациями. Вначале, когда о пользователе ничего не известно, ему могут быть отображены наиболее популярные объекты электронного ресурса, или предложены заранее созданные подборки по категориям.

По мере получения большей информации о пользователе перед рекомендательной системой встает задача проинформировать клиента о товаре, который может быть интересен ему [1]. Это увеличивает эффективность реализации продуктов. Так, по данным компании McKinsey за 2013 год, 35% купленных товаров в магазине Amazon и 75% просмотренного контента в сервисе Netflix было предоставлено пользователям через рекомендательные алгоритмы. Такой прирост спроса ведет к стремительному распространению систем рекомендаций, применению различных концепций при их разработке и внедрении в электронный ресурс.

1.1 Характеристики рекомендательных систем

Поскольку область применения рекомендательных систем достаточно широка, для описания каждой системы используется ряд характеристик [2]:

– Предмет рекомендации – характеризует тип анализируемой информации. Это может быть медиаконтент, заведения, рабочие места, товары, новости и т. п. Предмет рекомендации определяет особенности построения системы. Например, рекомендация новостей требует учета их

актуальности и большого количества вычислений в реальном времени, в то время как для рекомендации фильмов эти свойства не критичны.

– Цель – целью рекомендаций может быть, помимо продажи рекомендованного продукта, информирование пользователя о чем-либо, предоставление персональных предложений и т. д.

– Контекст – под контекстом понимаются условия, в которых пользователь подучает информацию. Например, рекомендации могут быть скорректированы в зависимости от времени суток, погоды, устройства. Например, видеохостинг YouTube может предлагать различные видео на смартфонах и телевизорах.

– Уровень персонализации – различаются неперсонализированные, полуперсонализированные и персонализированные рекомендации. Неперсонализированные рекомендации представляют собой общие списки продуктов, ранжированные по популярности, темам и т. п. При использовании полуперсонализированных предложений списки продуктов составлены с учетом их популярности в группах пользователей по возрасту, городу, стране, используемому устройству. Персонализированные рекомендации составляются на основе имеющихся данных о конкретном пользователе.

– Интерфейс – определяет тип получения входных и выдачи выходных данных. Входные данные могут собираться явным и неявным образом. При сборе входных данных явным образом пользователю предлагается поставить оценку по определенной шкале, или добавить продукт в избранное. Неявный сбор данных осуществляется системой на основе взаимодействия пользователя с системой; например, музыкальные стриминговые сервисы могут учитывать прослушивание песни до конца, пропуск песни, прослушивание первых 30 секунд и др. Что касается выдачи выходных данных, то они могут отображаться по-разному в пределах одного ресурса. Рекомендуемые продукты могут быть неявно интегрированы в сайт или показываться в специальном списке; а также может быть объяснено, почему продукт рекомендуется, либо объект предлагается без объяснений.

– Алгоритмы – существует множество различных алгоритмов для рекомендательных систем, однако они основаны на нескольких базовых подходах. Подходы выбираются в зависимости от требований к рекомендациям, имеющихся вычислительных ресурсов и т. д. Они будут рассмотрены в данной работе в отдельном разделе.

1.2 Требования к рекомендательным системам

В практической деятельности при разработке рекомендательной системы для нее определяется ряд требований. Так, с точки зрения реакции пользователя рекомендуемые объекты должны обладать следующими качествами [3]:

– Релевантность – в первую очередь, рекомендации должны соответствовать ожиданиям пользователя. Только в этом случае он будет чувствовать доверие к рекомендательной системе. Возможно, пользователь пойдет системе навстречу и будет более охотно и искренне выставлять оценки.

– Новизна – предлагаемые объекты должны быть новыми для пользователя. Если будут предложены уже просмотренные объекты, или каким-либо другим образом пользователю будет хорошо известен объект, он может перестать следить за рекомендациями.

– Сенсационность – если пользователю предложить для ознакомления объект, который не совсем относится к предпочтительным категориям, и он ему понравится, пользователь воспримет это как открытие для себя новой категории или жанра. Таким образом, рекомендательная система дает пользователю возможность попробовать что-нибудь новое и необычное для него.

– Разнообразие – система должна предлагать не только самые подходящие объекты, но и таковые из смежных категорий. Поскольку пользователь может перестать предпочитать некоторую конкретную

категорию, и он может захотеть сменить или расширить круг своих предпочтений.

– Прозрачность [1] – пользователь больше доверяет рекомендациям, если ему понятно, как они были получены. В противном случае пользователь будет полагать, что сервис стремится им манипулировать, навязывать отдельные продукты. Чтобы этого избежать, система должна уметь противодействовать влиянию поддельных оценок на рекомендации.

1.3 Проблемы рекомендательных систем

В процессе работы рекомендательных систем возникают различные проблемы, связанные с особенностями поведения пользователей и работы алгоритмов. Рассмотрим основные из них:

– Недостаток информации [3] – данная проблема может проявляться по-разному в зависимости от конкретной системы. Так, в системе, основанной на алгоритме коллаборативной фильтрации, при работе над новым объектом появляется проблема «холодного старта», когда за неимением достаточного числа оценок система не может дать точных рекомендаций. Данную проблему можно исправить, используя алгоритм фильтрации на основе содержимого. Однако такой системе требуется описательная информация о рекомендуемых объектах, сбор которой также может быть проблематичен.

– «Пузырь фильтров» [4] – система может предоставлять лишь контент, собранный на основе истории просмотров либо покупок пользователя, не добавляя иной продукции из смежных категорий. Таким образом, область рекомендаций объектов может быть замкнута. Это ведет к негативным последствиям. Например, у пользователя сложится впечатление, что его позиция ограничивается системой. В результате он может покинуть ресурс.

– Конфиденциальность [5] – рекомендательные алгоритмы могут распознавать факторы, которые могли быть слишком конфиденциальными

для пользователя. Например, известен случай, когда система на основе предпочтений пользователя могла определять беременность и его срок. Открытым остается вопрос степени конфиденциальности информации, анализ которой позволительно осуществлять.

– Защита от атак [3] – рекомендательные системы могут прогнозировать ложные и необъективные рейтинги, основываясь на оценках, умышленно заниженных или завышенных недобросовестными пользователями. Однако оценки могут быть завышены и непреднамеренно. Например, когда фильм выходит в прокат, в первое время рейтинг может быть завышен из-за большого числа оценок фанатов.

1.4 Классификация рекомендательных систем

Рекомендательные системы разрабатываются с использованием ряда моделей, основанных на различном понимании того, по каким признакам объект должен считаться предпочтительным. Ниже описаны основные подходы к составлению рекомендаций [3]:

1) Коллаборативная фильтрация – 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 ней выделяют два аналогичных по принципу работы подхода: коллаборативную фильтрацию по пользователям и по товарам.

– Методы, основанные на моделях – предполагают проведение предварительных операций над матрицей оценок, создание на ее основе описательной модели, чтобы выявить скрытые факторы, влияющие на рейтинги пользователей. Преимуществом данных методов 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 относится семейство различных видов разложений матрицы, наивный байесовский классификатор, методы на основе марковских моделей и др.

2) Фильтрация на основе содержимого – в таком подходе 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.

3) Системы, основанные на знаниях – такие системы основываются на явных запросах пользователя, поскольку они предназначены для рекомендации товаров, которые покупаются нечасто (например, недвижимость, транспорт и т. п.). В таких ситуациях для определения подходящего товара невозможно просматривать историю покупок пользователя либо отталкиваться от выбора других клиентов. Разработчик системы должен вместе с экспертами создать в ней базу знаний, по которой будут выбираться критерии, определяющие подходящий результат под конкретный запрос пользователя.

4) Демографические рекомендательные системы – выполняют задачу подбора товаров, основываясь на принадлежности пользователя к той или иной демографической группе, например, по признаку возраста, пола, местонахождения, или устройства, с которого он посещает сайт. Построение такой системы может быть сведено к задаче классификации товаров по соответствующим группам пользователей. В большей мере демографические рекомендательные системы используются не отдельно, а в составе гибридных систем рекомендаций.

5) Гибридные рекомендательные системы – выданные такой системой рекомендации представляют собой комбинацию результатов систем прочих типов. Результаты могут быть получены посредством реализации различных алгоритмов в одном модуле (рисунок 1), параллельной работы различных модулей и их комбинирования (например, через получение взвешенной суммы) (рисунок 2), или путем каскадного применения различных систем (рисунок 3).

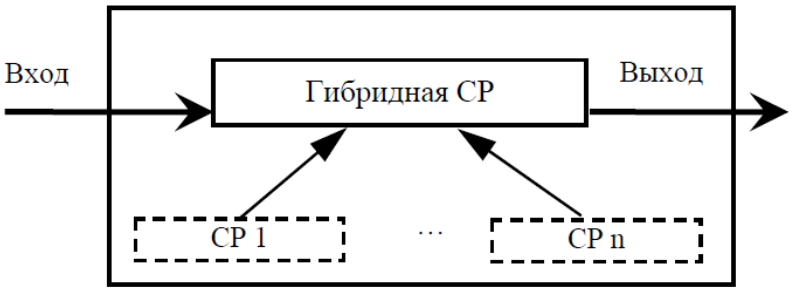

Рисунок 1 – Модульная организация системы

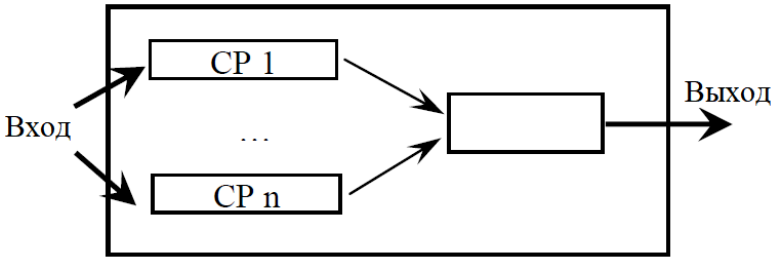

Рисунок 2 – Параллельная организация системы

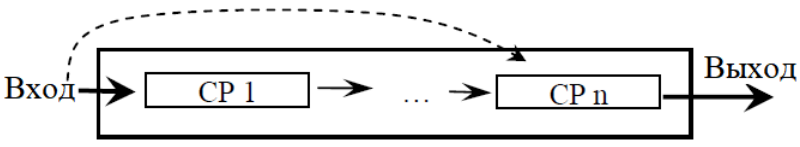

Рисунок 3 – Каскадная (конвейерная) организация системы

2 Коллаборативная фильтрация

2.1 Постановка задачи

Пусть рекомендательной системе доступно множество пользователей U и множество объектов I. Объектами могут быть, например, товары. Пусть некоторый пользователь $u \in U$ выставил некоторому товару $i \in I$ рейтинг $r_{u,i}$, которая может принимать вещественные значения от 0 до 5.

Из этих данных можно построить матрицу оценок, в которой каждому пользователю u соответствует одна строка, а каждому объекту i – один столбец. Тогда элементы матрицы будут представлять собой рейтинг $r_{u,i}$, если пользователь u поставил оценку объекту i, либо 0, если оценка не выставлена.

В алгоритме также используется средняя оценка пользователя и средняя оценка товара. Обозначим их соответственно $\overline{r_u}$ и $\overline{r_i}$.

От программы требуется выполнить две задачи [6]:

1) предсказать приблизительную оценку каждого пользователя каждому товару и заполнить полученными оценками матрицу;

2) на основе вычисленных оценок получить ранжированный список объектов, которые будут рекомендованы пользователю.

2.2 Описание алгоритма

В системах коллаборативной фильтрации, основанных на работе с памятью, различают два подхода к вычислению приблизительной оценки:

1) фильтрация на основе сходства пользователей – для получения рейтинга пользователя товару учитываются оценки, выставленные этому товару похожими пользователями;

2) фильтрация на основе сходства объектов – для предсказания рейтинга учитываются оценки, выставленные рассматриваемым пользователем объектам, похожим на рассматриваемый товар.

В обоих подходах в первую очередь необходимо определить функцию сходства между двумя пользователями (при использовании первого подхода), либо между двумя объектами (во втором подходе). В качестве таких мер близости могут использоваться многие функции. В данной работе рассмотрим три основные формулы: косинусное сходство, коэффициент корреляции Пирсона и нормированное косинусное сходство.

При использовании косинусного сходства близость двух пользователей вычисляется как косинус угла между векторами, соответствующими их строкам в матрице оценок. Такая мера дает наибольшую близость, если оценки пользователей соответствующим объектам пропорциональны. Так, косинусное сходство пользователей u и v вычисляется формулой (1):

$$sim(u,v) = \frac{\sum_{i \in I} r_{u,i} r_{v,i}}{\sqrt{\sum_{i \in I} r_{u,i}^{\ 2}} \sqrt{\sum_{i \in I} r_{v,i}^{\ 2}}} \quad (1)$$

Коэффициент корреляции Пирсона отражает степень линейной зависимости между двумя центрированными векторами. Близость зависит от того, насколько высоко или низко пользователь оценил товар относительно своих остальных оценок. Для векторов пользователей u и v формула коэффициента корреляции (2) принимает вид:

$$sim(u,v) = \frac{\sum_{i \in I} \left(r_{u,i} - \overline{r}_u\right)\left(r_{v,i} - \overline{r}_v\right)}{\sqrt{\sum_{i \in I} \left(r_{u,i} - \overline{r}_u\right)^2} \sqrt{\sum_{i \in I} \left(r_{v,i} - \overline{r}_v\right)^2}} \quad (2)$$

Нормированное косинусное сходство [7] вычисляет сходство пользователей аналогично косинусной сходимости, однако используются не вектора оценок пользователей, а вектора отклонений их оценок от среднего рейтинга объекта. Таким образом, чем более одинаково оценки пользователей

для некоторого объекта отклоняются от «общепринятых» оценок объекта, тем большее сходство между пользователями показывает данная функция. Это видно из формулы (3):

$$sim(u,v) = \frac{\sum_{i \in I} \left(r_{u,i} - \overline{r_i}\right)\left(r_{v,i} - \overline{r_i}\right)}{\sqrt{\sum_{i \in I} \left(r_{u,i} - \overline{r_i}\right)^2} \sqrt{\sum_{i \in I} \left(r_{v,i} - \overline{r_i}\right)^2}} \quad (3)$$

Следующим этапом после выбора функции сходства необходимо определить для каждого пользователя $u \in U$ множество близких пользователей $K_u \subset U$, «соседей», из оценок которых будет складываться предполагаемая оценка объекта. Для этого используются следующие подходы [8]:

– установка порога – соседом считается тот пользователь, мера близости с которым превышает определенное значение;

– поиск k ближайших соседей – множество состоит из k пользователей с наибольшим сходством, где k – заранее выбранная константа.

Выбор между тем или иным подходом определяется из того, что приоритетнее для вычисления оценки, качество (первый подход) или количество (второй подход) [2].

Имея множество близких пользователей, мы можем найти предполагаемую оценку $\hat{r}_{u,i}$ по формуле (4):

$$\hat{r}_{u,i} = \overline{r_u} + \frac{\sum_{k \in K_u} \left(r_{k,i} - \overline{r_k}\right) sim(u,k)}{\sum_{k \in K_u} \left|sim(u,k)\right|} \quad (4)$$

Аналогично устроена фильтрация на основе сходства объектов. В ней вычисляется мера близости между двумя объектами $sim(i, j)$ по формулам,

аналогичным формулам (1), (2) и (3); для каждого объекта $i \in I$ определяется множество близких объектов $K_i \subset I$, и вычисляется предполагаемая оценка $\hat{r}_{u,i}$ по формуле (5):

$$\hat{r}_{u,i} = \overline{r}_i + \frac{\sum_{k \in K_i} \left(r_{u,k} - \overline{r}_k \right) sim(i,k)}{\sum_{k \in K_i} \left| sim(i,k) \right|} \quad (5)$$

2.3 Программная реализация

Для демонстрации системы коллаборативной фильтрации была написана программа, прогнозирующая рейтинг фильма, на языке программирования Python с использованием библиотеки NumPy. Программа вычисляет рейтинг, основываясь на оценках пользователя к другим фильмам.

Ее работа проходит в следующих этапах:

– составление матрицы «пользователь-фильм»;

– вычисление среднего рейтинга пользователя;

– заполнение матрицы схожести пользователей;

– для каждого пользователя вычисляется его рейтинг к каждому фильму по формуле (4) или (5), и по завершению работы программы выводится заполненная матрица оценок;

– на основе полученной матрицы программа составляет ранжированный список фильмов, рекомендованных пользователю.

Для создания матрицы был выбран набор данных MovieLens [9], предоставляемый исследовательской лабораторией GroupLens Университета Миннесоты. Данный набор содержит 100836 рейтингов к 9724 фильмам от 610 пользователей. Рейтинги хранятся в файле типа csv.

Программа состоит из трех файлов: Extract_data.py (приложение А), Recommendations.py (приложение Б) и Print_top.py (приложение В).

В программе Extract_data.py извлекается информация из csv-файла в виде матрицы. В ней пока в каждой строке записаны идентификационный номер (ID) пользователя, ID фильма и рейтинг. Для выполнения алгоритма строится новая матрица из n строк и m столбцов, где n – количество пользователей, m – количество оцененных фильмов. Также для каждого фильма был определен новый ID для его идентификации только среди тех фильмов, у которых есть хотя бы одна оценка.

Просматривая каждую строку с пользователем, фильмом и рейтингом, программа сохраняет оценку в элементе матрицы, строка и столбец которого соответствуют пользователю и рейтингу. Дальнейшая работа программы будет проходить с полученной матрицей.

Для проверки рекомендательного алгоритма входные данные нужно будет распределить в обучающую матрицу и тестовую [10]. Разделение данных на обучающую и тестовую выборки является распространенным методом вычисления ошибки прогнозирования. При нем часть имеющихся данных скрывается и собирается в тестовую часть. Оставшиеся данные представляют собой учебную выборку. Выполнение основного алгоритма будет производиться именно над учебной выборкой. После выполнения вычислений программа, помимо изначально неизвестных данных, вычислит данные, которые были преднамеренно убраны. Это дает возможность проверить, насколько выходные данные программы соответствуют действительности. Для этого вычисляется отклонение вычисленных результатов от фактических показателей в тестовой выборке.

Разделение данных выполняется в основной программе Recommendations.py. Тестовая выборка составляется случайным образом так, чтобы в нее было перенесено 10% рейтингов от каждого пользователя.

Далее объявляется функция для вычисления среднего арифметического ненулевых элементов вектора. С ее помощью составляются вектора со средними рейтингами каждого пользователя и средними рейтингами каждого фильма.

Чтобы вычислить сходство пользователей и фильмов, была добавлена функция similarity(matr, type), возвращающая матрицу сходства строк. Она может в зависимости от значения параметра type вычислять косинусное сходство, коэффициент корреляции Пирсона и нормированное косинусное сходство, используя методы библиотек Scikit-learn и NumPy. С ее помощью составляются матрицы сходства пользователей и фильмов.

В программе также объявлена функция k_nbr(amount, usr, mov, type), которая по ID пользователя usr и фильма mov возвращает список из ID ближайших соседей пользователя в количестве amount, оценивших фильм mov. Если указать параметр type как «movie», то она вернет список из amount фильмов, самых похожих на фильм mov, оцененных пользователем usr.

Функция rmse(train, test) вычисляет среднеквадратическую ошибку между элементами тестовой матрицы test и предсказанными оценками обучающей матрицы train по следующей формуле (6):

$$RMSE = \sqrt{\frac{1}{|T|}\sum_{r_{u,i}\in T}\left(r_{u,i} - \hat{r}_{u,i}\right)^2}, \qquad (6)$$

где T – множество оценок, отнесенных к тестовой выборке.

Основной функцией предсказания оценок является подпрограмма recommend(type, printmode). Параметр type отвечает за то, выполнять фильтрацию по пользователям или по фильмам, а printmode – за вывод промежуточных результатов, помимо среднеквадратической ошибки.

Проходя в цикле по каждому элементу матрицы оценок и используя заранее определенные матрицы сходств, вектора средних оценок и функцию k_nbr, подпрограмма recommend вычисляет предполагаемый рейтинг по формуле (4) для пользователей и (5) для фильмов.

Из полученной матрицы оценок построим матрицу рекомендуемых фильмов top, состоящую из n строк и top_size столбцов, где top_size – переменная, содержащая длину ранжированного списка. В ней каждая строка

представляет заполненный для каждого пользователя массив ID фильмов, отсортированный по убыванию рейтингов.

Следующим этапом является составление из матрицы top списка рекомендуемых фильмов. Для этого в наборе данных имеется второй файл типа csv, каждая строка которого содержит пару «ID фильма – название фильма». Считывая из каждой строки матрицы top номер фильма (который ранее был изменен), получим из него первоначальный ID, и по нему найдем название фильма. Затем выведем его в консоль, получая для каждого пользователя названия фильмов которые ему рекомендованы (рисунок 4).

Рисунок 4 – Демонстрация вывода программы

2.4 Результат работы программы.

При разном количестве k близких пользователей, просматриваемых для вычисления оценок, программа может давать различные прогнозы. Рассмотрим изменение среднеквадратической ошибки предсказания рейтингов на алгоритме коллаборативной фильтрации при различных значениях k (таблица 1).

Таблица 1 – Результаты эксперимента

Число «соседей» k	Среднеквадратическая ошибка *RMSE*
20	0.868993825551664
25	0.8669792384035222
30	0.8670085468166983
35	0.8672531989069422
40	0.8676652974264537

Ниже для сравнения на приведен рейтинг алгоритмов-победителей конкурса Netflix Prize, проводимого компанией Netflix в 2007-2009 гг. (рисунок 5) [11]. Таким образом, значение 0,86-0,87 можно считать допустимым для среднеквадратической ошибки.

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59

Рисунок 5 – Рейтинг алгоритмов со значением *RMSE*

3 Транзитивные ассоциативные сети

3.1 Описание алгоритма

Стандартные алгоритмы коллаборативной фильтрации работают по принципу рекомендации тех объектов, которые пользователь не оценивал, но они понравились пользователям, похожим на него. Однако они не учитывают, что у тех похожих пользователей также есть «соседи», и их предпочтения могут сыграть роль в рекомендациях для пользователя, рассмотренного в начале. Чтобы было возможно рекомендовать товары по признаку транзитивного сходства пользователей (сходства за счет общего «соседа»), был предложен иной подход, основанный на построении транзитивных связей между пользователями и объектами [7].

Предположим, что имеются пользователи u_1, u_2, u_3 и товары i_1, i_2, i_3, i_4, и их матрица оценок состоит из бинарных значений и выглядит следующим образом:

$$\begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \end{array} \begin{array}{c} \begin{array}{cccc} i_1 & i_2 & i_3 & i_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

При использовании метода коллаборативной фильтрации похожими будут считаться пользователи u_1 и u_2, а также u_2 и u_3. В то же время система посчитает интересы u_1 и u_3 противоположными, и в связи с этим, например, товар i_1 не будет предложен пользователю u_1.

Рассмотрим транзитивность сходства пользователей на графе покупок (рисунок 6). В нем пользователи и товары представляют собой вершины. Вершина пользователя соединена с вершиной товара дугой, если пользователь приобрел данный товар.

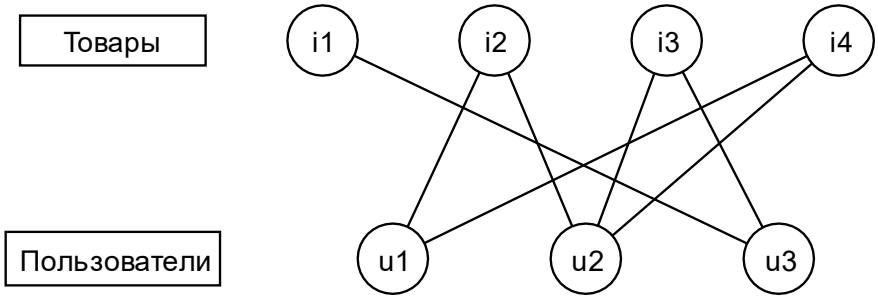

Рисунок 6 – Граф пользователей и товаров

В графе рекомендация товара определяется количеством путей между их вершинами. Так, от пользователя u_1 до товара i_3 существует два пути: $u_1 - i_2 - u_2 - i_3$ и $u_1 - i_4 - u_2 - i_3$. Оба этих пути длины 3 будут учтены системой коллаборативной фильтрации, так как связь между пользователем и товаром определяется лишь через одного пользователя u_2. Но транзитивная связь u_1 и u_3 (например, через путь $u_1 - i_2 - u_2 - i_3 - u_3$) учтена не будет, а значит, товара i_1 не будет в рекомендациях пользователю u_1.

При использовании транзитивных ассоциативных сетей в графе рассматриваются все пути длины, не превышающей некоторое значение M. При этом чем больше длина пути от пользователя до товара, тем меньше будет рекомендательная сила данного товара для него. Чтобы этого добиться, можно каждый путь длины x прибавлять со степенью удаленности ρ^x, где $\rho \in (0,1)$. Таким образом, итоговая рекомендательная сила товара i_1 пользователю u_1, будет вычислена как $\rho^5 + \rho^5 = 2 \cdot 0,5^5 = 0,0625$ (для $\rho = 0,5$), поскольку между их вершинами существует два пути длиной 5.

Чтобы вычислить количество путей от пользователя к товару, можно построить матрицу достижимости $A(x)$ для путей длины x. Элементы матрицы будут представлять собой количество путей длины x от пользователя к товару. Такая матрица вычисляется по формуле (7):

$$A(x) = \begin{cases} A, & x = 1 \\ A \cdot A^T \cdot A(x-2), & x = 3,5,7... \end{cases} = \left(A \cdot A^T\right)^{\frac{x-1}{2}} \cdot A \qquad (7)$$

Таким образом, для вычисления общего количества путей от пользователя к товару, длина которых не превышает M (отметим, что длина пути от пользователя к товару всегда нечетна), нужно просуммировать значения $A(1), A(3), \ldots A(M)$, учитывая их вес ρ^x. Тогда получим функцию $\alpha(M)$ которая представляет собой матрицу рекомендательных сил товаров и которую можно найти по формуле (8):

$$\alpha(M) = \sum_{x=1}^{\frac{M+1}{2}} \rho^{2x-1} \cdot A(2x-1) = A \cdot \sum_{x=1}^{\frac{M+1}{2}} \rho^{2x-1} \cdot \left(A \cdot A^T\right)^{\frac{2x-1-1}{2}} \qquad (8)$$

3.2 Программная реализация

Воспользуемся матрицей, созданной ранее для системы коллаборативной фильтрации в программе Extract_data.py. Чтобы ее можно было использовать в программе для построения транзитивной ассоциативной сети, сделаем ее значения бинарными. Для этого все ненулевые значения матрицы приравняем к 1.

Задача программы состоит в том, чтобы:

1) при заданных M и ρ вычислить матрицу по функции $\alpha(M)$;

2) на основе матрицы рекомендательных сил составить ранжированный список фильмов, рекомендованных пользователю.

Программа будет располагаться в файле Transitive_networks.py (приложение Г). В начале ее работы вычисляется матрица $A \cdot A^T$, которая имеет размер $n \times n$, где n – количество пользователей. Ее элементы характеризуют количество общих товаров между пользователями, иначе говоря, количество путей длины 2 между пользователями.

После этого вызывается подпрограмма alpha(ro, m), которая возвращает матрицу рекомендательных сил по функции $\alpha(M)$ при M и ρ, инициализированных ранее в программе. Ее работа состоит в том, что она выполняет цикл по x от 1 до M с шагом 2. В каждой итерации цикла программа вычисляет компонент $\rho^x \cdot A(x)$ по формуле (9) при $x = 3, 5, \ldots M$, прибавляя его в конце цикла к переменной суммы.

$$\rho^x \cdot A(x) = \rho^2 \cdot A \cdot A^T \cdot \left(\rho^{x-2} \cdot A(x-2)\right) \quad (9)$$

Получив матрицу рекомендательных сил, построим матрицу рекомендуемых фильмов. Для этого инициализируем переменную top_size, которая определяет длину ранжированного списка. Создадим матрицу top из n строк и top_size столбцов, в которой каждая строка представляет собой заполненный для каждого пользователя массив ID фильмов, отсортированный по убыванию рекомендательной силы. Затем, с помощью описанной в предыдущей главе программы print_top.py, выведем по этой матрице список фильмов (рисунок 7).

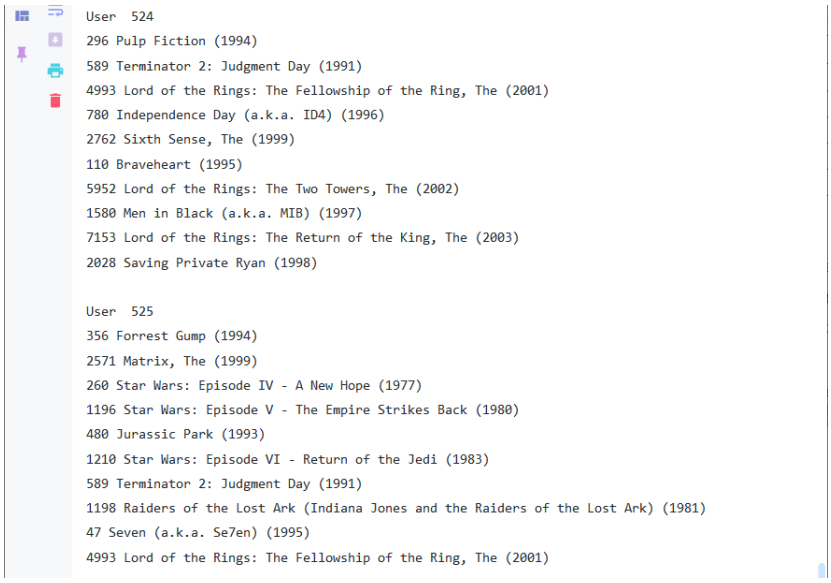

Рисунок 7 – Демонстрация вывода программы

В отличие от системы коллаборативной фильтрации, данная программа работает с бинарной матрицей, и рекомендация фильма определяется тем, сколько путей и какой длины существует от вершины пользователя до вершины фильма. Если от пользователя фильм достигается через многих других пользователей и на малом расстоянии, фильм имеет большую рекомендательную силу. Таким образом, данный алгоритм вычисляет (без учета оценок) самый распространенный фильм среди пользователей, в первую очередь близких.

ЗАКЛЮЧЕНИЕ

В данной курсовой работе были изучены основные теоретические сведения о рекомендательных системах. Составлено описание отличительных черт различных систем рекомендаций, приведены основные требования к рекомендуемым объектам, рассмотрены проблемы, вызванные особенностями работы систем. Были описаны классы алгоритмов, лежащих в основе рекомендательных систем.

Далее в работе были изучены и описаны методы создания системы коллаборативной фильтрации на основе работы с памятью. Данная система была реализована на языке программирования Python. Также были описаны результаты ее работы.

Помимо этого были даны сведения о рекомендательных системах, работающих на основе транзитивных ассоциативных сетей. Приведены принципы их работы и используемый в них алгоритм. Для рассмотренного алгоритма была составлена и описана программная реализация и продемонстрирована ее работа.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Анатомия рекомендательных систем. Часть первая / Блог компании ГК ЛАНИТ / Хабр. – URL: https://habr.com/ru/company/lanit/blog/420499/ (дата обращения 15.12.2019).

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

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

4 Пузырь фильтров — Википедия. – URL: https://ru.wikipedia.org/wiki/%D0%9F%D1%83%D0%B7%D1%8B%D1%80%D1%8C_%D1%84%D0%B8%D0%BB%D1%8C%D1%82%D1%80%D0%BE%D0%B2 (дата обращения 17.12.2019).

5 Джонс М. Рекомендательные системы: Часть 1. Введение в подходы и алгоритмы. 29.04.2014. URL: https://www.ibm.com/developerworks/ru/library/os-recommender1/index.html (дата обращения 14.12.2019).

6 Пятикоп Е.Е. Исследование метода коллаборативной фильтрации на основе сходства элементов // Научные работы ДонГТУ. Серия «Информатика, кибернетика и вычислительная техника». – 2013. – №2. – С. 109-114.

7 Рекомендательные системы [Электронный ресурс] // Business Data Analytics: сайт. – URL: http://www.businessdataanalytics.ru/download/recommendationsystems.pdf (дата обращения 25.10.2019).

8 Гомзин А. Г. Системы рекомендаций: обзор современных подходов / А. Г. Гомзин, А. В. Коршунов // Труды Института системного программирования РАН. М.: Институт системного программирования РАН, 2012 – С. 401-417.

9 MovieLens Latest Datasets | GroupLens – URL: https://grouplens.org/datasets/movielens/latest/ (дата обращения 11.11.2019).

10 Коэльо Л. П. Построение систем машинного обучения на языке Python. 2-е издание / Л. П. Коэльо, В. Ричарт – М.: ДМК Пресс, 2016. – 302 с.

11 Netflix Prize: View Leaderboard. – URL: https://www.netflixprize.com/leaderboard.html (дата обращения 16.12.2019).

ПРИЛОЖЕНИЕ А

Содержимое файла Extract_data.py

```
import numpy as np

ratings = (np.loadtxt('ml-latest-small/ratings.csv',
                      delimiter=',', skiprows=1, usecols=(0, 1, 2)))

# Определим для фильмов новые id - чтобы нумеровались только
# те фильмы, которые есть в рейтингах:
# 1. Создаем массив уникальных фильмов
movies = np.unique(ratings[:, 1].astype(int))

# 2. Заменяем id фильмов в ratings на их индекс в массиве movies
for i, movieid in enumerate(ratings[:, 1]):
    movieid_for_matr = np.argwhere(movies == movieid)[0][0] + 1
    ratings[i, 1] = movieid_for_matr

# Найдем количество пользователей
userssize = (np.unique(ratings[:, 0].astype(int))).size

# создаем матрицу рейтингов Пользователь-Фильм
UM_matrix = np.zeros((userssize, movies.size))
for rating_string in ratings:
    my_user = rating_string[0].astype(int)
    my_movie = rating_string[1].astype(int)
    UM_matrix[my_user - 1, my_movie - 1] = rating_string[2]
```

ПРИЛОЖЕНИЕ Б

Содержимое файла Recommend.py

```
import numpy as np
import sklearn.metrics.pairwise as pairs
from sklearn.metrics import mean_squared_error
from math import sqrt

from Extract_data import UM_matrix


# Разделим данные на обучающие и тестовые
# Перенесем 10% рейтингов пользователя в тестовую выборку
testpart = 0.1
UM_matrix = UM_matrix[:, :1000]
test_matrix = np.zeros_like(UM_matrix)
train_matrix = UM_matrix.copy()
np.random.seed(1234)
for user in range(train_matrix.shape[0]):
    count_nz = np.count_nonzero(train_matrix[user])
    nz_choice = np.nonzero(train_matrix[user])[0]
    test_choice = np.random.choice(nz_choice, size=int(testpart * count_nz),
replace=False)
    test_matrix[user, test_choice] = train_matrix[user, test_choice]
    train_matrix[user, test_choice] = 0


# Функция средней оценки в ненулевых элементах матрицы
def mean_nonzero(matr, _axis):
    numnonzero = np.count_nonzero(matr, axis=_axis)
    numnonzero[numnonzero == 0] = 1
    stringsum = np.sum(matr, axis=_axis)
    return np.true_divide(stringsum, numnonzero)


# Находим средний рейтинг каждого пользователя по строкам
# и фильма по столбцам
mean_userrating = mean_nonzero(train_matrix, 1)

mean_movierating = mean_nonzero(train_matrix, 0)


# Функция схожести
def similarity(matr, type='pearson'):
    if type == 'cosine':
        return pairs.cosine_similarity(matr)
    if type == 'pearson':
        return np.corrcoef(matr)
    if type == 'adjusted_cosine_users' or 'adjusted_cosine_movies':
        mr = np.empty(0)
        if type == 'adjusted_cosine_users':
            mr = mean_movierating
        elif type == 'adjusted_cosine_movies':
            mr = np.array([mean_movierating]).T
        diversities = matr - mr
        return pairs.cosine_similarity(diversities)
```

```
# Рассчитаем схожесть пользователей и схожесть фильмов друг с другом
user_similarity = similarity(train_matrix, type='pearson')

movie_similarity = similarity(train_matrix.T, type='pearson')


# Вектора пользователей, отсортированные от непохожих до "соседей"
user_nearest = user_similarity.argsort(axis=1)

movie_nearest = movie_similarity.argsort(axis=1)

k = 25


# Функция для списка из k ближайших соседей, смотревших фильм
def k_nbr(amount, usr, mov, type='user'):
    k_list = []
    i = -1

    if type == 'user':
        neighbours = user_nearest[usr]
        while len(k_list) < amount and -i < neighbours.size:
            if train_matrix[neighbours[i], mov] > 0:
                k_list.append(neighbours[i])
            i -= 1

    elif type == 'movie':
        neighbours = movie_nearest[mov]
        while len(k_list) < amount and -i < neighbours.size:
            if train_matrix.T[neighbours[i], usr] > 0:
                k_list.append(neighbours[i])
            i -= 1

    return k_list


# Функция вычисления среднекватратического отклонения
def rmse(train, test):
    test_nz_choice = np.nonzero(test.flatten())
    mse = mean_squared_error(train.flatten()[test_nz_choice],
test.flatten()[test_nz_choice])
    return sqrt(mse)

top_size = 10

# Функция вычисления оценки
def recommend(type='user', printmode=True):
    if type == 'user':
        train_matrix_copy = train_matrix.copy()
        for usrid in range(train_matrix_copy.shape[0]):
            for movid in range(train_matrix_copy.shape[1]):
                if train_matrix_copy[usrid, movid] == 0:
                    k_nearest = k_nbr(k, usrid, movid, 'user')
                    rate = mean_userrating[usrid]

                    sum1 = 0
                    sum2 = 0
                    for nbr in k_nearest:
                        sum1 += (train_matrix_copy[nbr, movid] -
mean_userrating[nbr]) * user_similarity[usrid, nbr]
                        sum2 += abs(user_similarity[usrid, nbr])
```

```
                    rate += sum1/sum2 if sum2 > 0 else 0.0
                    train_matrix_copy[usrid, movid] = rate

            if printmode:
                # Вывод результата для каждого пользователя
                print('\nUser ', usrid)
                print('Ratings: ', train_matrix_copy[usrid])

                test_nz_choice = np.nonzero(test_matrix[usrid])
                if len(test_nz_choice[0]) > 0:
                    print('Predicted for test: ', train_matrix_copy[usrid,
test_nz_choice])
                    print('Real in test: ', test_matrix[usrid, test_nz_choice])
                    print('RMSE for user ', usrid, ' = ',
rmse(train_matrix_copy[usrid], test_matrix[usrid]))
        print('\nUser-based CF RMSE = ', rmse(train_matrix_copy, test_matrix))
        return train_matrix_copy

    elif type == 'movie':
        train_matrix_T = train_matrix.T.copy()

        for movid in range(train_matrix_T.shape[0]):
            for usrid in range(train_matrix_T.shape[1]):
                if train_matrix_T[movid, usrid] == 0:
                    k_nearest = k_nbr(k, usrid, movid, 'movie')
                    rate = mean_movierating[movid]
                    sum1 = 0
                    sum2 = 0
                    for nbr in k_nearest:
                        sum1 += (train_matrix_T[nbr, usrid] - mean_movierating[nbr])
* movie_similarity[movid, nbr]
                        sum2 += abs(movie_similarity[movid, nbr])
                    rate += sum1/sum2 if sum2 > 0 else 0.0
                    train_matrix_T[movid, usrid] = rate
            if printmode:
                # Вывод результата для каждого фильма
                print('\nMovie ', movid)
                print('Ratings: ', train_matrix_T[movid])
                test_nz_choice = np.nonzero(test_matrix.T[movid])
                if len(test_nz_choice[0]) > 0:
                    print('Predicted for test: ', train_matrix_T[movid,
test_nz_choice])
                    print('Real in test: ', test_matrix.T[movid, test_nz_choice])
                    print('RMSE for movie ', movid, ' = ',
rmse(train_matrix_T[movid], test_matrix.T[movid]))
        print('\nItem-based CF RMSE = ', rmse(train_matrix_T, test_matrix.T))
        return train_matrix_T.T


user_cf = recommend('user', True)

top = train_matrix.argsort(axis=1)
top = top[:, ::-1]
top = top[:, :top_size]
```

ПРИЛОЖЕНИЕ В

Содержимое файла Print_top.py

```
import pandas as pd
from Recommendations import top
# from Transitive_network import top
from Extract_data import movies, startTimer


movie_df = pd.read_csv('ml-latest-small/movies.csv', delimiter=',', header=0,
usecols=['movieId', 'title'])

for usrid in range(top.shape[0]):
    print('\nUser ', usrid)
    for movid in range(top.shape[1]):
        mov_tuple = movie_df[movie_df['movieId'] == movies[top[usrid, movid]]]
        print(movies[top[usrid, movid]], (mov_tuple['title'].tolist())[0])
```

ПРИЛОЖЕНИЕ Г

Содержимое файла Transitive_networks.py

```
import numpy as np
from extract_data import UM_matrix

A = UM_matrix.copy()
A[A > 0] = 1

aat = np.matmul(A, A.T)


def alpha_user_item(ro, m):
    x = 1
    rox = ro
    Ax = A.copy()
    Wx = rox * Ax
    _alpha = Wx
    while x < m:
        rox *= ro * ro
        np.matmul(aat, Ax, Ax)
        Wx = rox * Ax
        _alpha += Wx
        x += 2
    return _alpha


alfa = alpha_user_item(0.5, 19)
alfa[A > 0] = 0

top_size = 10
top = alfa.argsort(axis=1)
top = top[:, ::-1]
top = top[:, :top_size]
```