

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

**ИССЛЕДОВАНИЯ АЛГОРИТМОВ КОЛЛАБОРАТИВНОЙ ФИЛЬТРАЦИИ И РАЗРАБОТКА РАСШИРЯЕМОЙ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ**

Работу выполнил Н. А. Задорожный

[СОДЕРЖАНИЕ](#_Toc33690)

[ВВЕДЕНИЕ 3](#_Toc33690)

[1 Моделирование предпочтения в рекомендательных системах 5](#_Toc33691)

[1.1 Рекомендательные системы 5](#_Toc33692)

[1.2 Построение рекомендательных систем 7](#_Toc33693)

[1.3 Задача коллаборативной фильтрации 8](#_Toc33694)

[1.3.1 Memory-based подход 9](#_Toc33695)

[1.3.2 Model-based подход 10](#_Toc33696)

[1.4 Модель предпочтения SVD 11](#_Toc33697)

1.5 Обратная связь в рекомендательных системах

[1.6 Учет контекста в рекомендательных системах 14](#_Toc33698)

[2 Матричная факторизация 16](#_Toc33699)

[2.1 Градиентный спуск 18](#_Toc33700)

[2.2 Метод чередующихся наименьших квадратов 19](#_Toc33701)

[3 Практическая реализация рекомендательной системы с неявной обратной связью через матричную факторизацию 20](#_Toc33702)

[3.1 Обрабатываемые данные 22](#_Toc33703)

3.2 Создание обучающего и тестового множеств для РС.

[3.3 Реализация ALS для рекомендательной системы с неявной обратной связью на языке Python 28](#_Toc33704)

[3.4 Ускоренный ALS 30](#_Toc33705)

[3.5 Оценивание рекомендательной системы 31](#_Toc33706)

[3.6 Пример выдачи рекомендаций 34](#_Toc33707)

[ЗАКЛЮЧЕНИЕ 39](#_Toc33708)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 40](#_Toc33709)

ВВЕДЕНИЕ

Рекомендательная система позволяет человеку обозначить свои вкусы и возвращает результаты, любопытные для него, базируясь на оценках других пользователей и своих предположениях. В отличие от поисковых систем, для получения от системы ответа не требуется четкого задания запроса. Вместо этого пользователю предлагается оценить некоторые объекты из коллекции, и на основании этих его оценок и сравнения их с оценками предыдущих пользователей строятся предположения о вкусах пользователя и возвращаются наиболее близкие к ним результаты, формируя для него персонализированную выдачу. В качестве набора оцениваемых объектов могут, к примеру, выступать:

а) каталог ссылок на веб-сайты;

б) лента новостей;

в) товары в электронном магазине;

г) коллекция книг в библиотеке.

В сферу применения подобных систем входят и ситуации, когда пользователь не ищет информацию по конкретному ключевому слову, а, к примеру, хочет получить список современных статей, похожих по тематике на те, которые он просматривал до этого. Рекомендательными системами занимаются с начала 90-х годов, однако серьезнейшим толчком к развитию этой темы явился конкурс Netflix Prize, организованный в 2006 г. компанией Netflix, одним из лидеров на американском рынке проката DVD. Компания предоставила участникам данные о 100 миллионах рейтингов, проставленных 480 тысячами пользователей 18 тысячам фильмов в течение 6 лет. Участник, показавший результаты на 10% лучше, чем их собственный движок рекомендаций, получал миллион долларов. И набор данных, и приз в десятки раз превышали аналоги. В итоге методы построения рекомендательных систем начали бурно развиваться, но задача оказалась сложной – несмотря на ажиотаж и жесточайшую конкуренцию, на выполнение условий у участников ушло три года.

Среди подходов выделяются три:

а) Content-based – строятся профили пользователей и объектов на основе анализа текстовой метаинформации объектов. Затем, с помощью некой меры близости, часто – коэффициента Пирсона, определяются объекты, близкие к профилю пользователя. К сожалению, несмотря на богатое наследие Information Retrieval, этот метод пока не удалось заставить работать эффективно. Например, строится система рекомендации интересных твитов в Twitter и явно видно, что вклад в качество системы от content-based компоненты сильно ниже.

б) Neighbourhood-based (NB) – подходящими объектами для пользователя user считаются объекты, высоко оцениваемые его «соседями», т.е. такими пользователями neib\_i, у которых оценки хорошо скоррелированы с user.

в) Model-based, matrix factorization (MF) – предполагается, что оценка пользователя определяется не очень большим числом (K=30-180) скрытых факторов (например, фильм азиатский или американский; арт-хаус либо блокбастер; современный либо старый т.п.). И задача здесь состоит в том, чтобы найти такие факторы, при выборе которых эта приближенная модель будет лучше всего приближать реальные пользовательские рейтинги. Для этого часто используется сингулярное разложение матриц или итеративное градиентное приближение. Один их таких алгоритмов будет подробно рассмотрен ниже. Считается, что MF лучше справляется с нахождением закономерностей в рамках коллекции в целом, и, будучи примененным в одиночку, дает более высокое качество, чем NB. Тем не менее, последний метод лучше находит локализованные особенности. К тому же, эти два метода хорошо сочетаются.

# 1 Моделирование предпочтения в рекомендательных системах

## 1.1 Рекомендательные системы

Рекомендательные системы — одно из наиболее популярных приложений интеллектуального анализа данных и машинного обучения в сфере интернетбизнеса. Рекомендательная система анализирует поведение пользователей интернет-сервиса, после чего может давать оценку предпочтения пользователем того или иного объекта рекомендаций. Объектами рекомендаций могут служить товары в интернет-магазине, набор разделов веб-сайта, медиаконтент, другие пользователи веб-сервиса. На основе предсказываемых рекомендательной системой предпочтений, поведение веб-сервиса к конкретному пользователю может меняться — такую функциональность принято называть персонализацией. На рисунке 1 представлен пример персональной рекомендации.

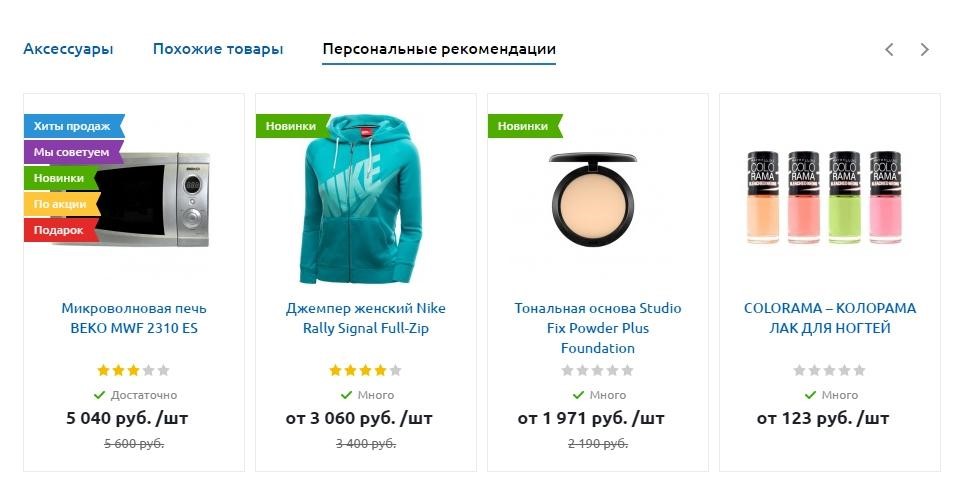


Рисунок 1 – Пример персональной рекомендации

На основании только что посмотренных пользователем объектов, либо используя данные предоставленные пользователем о его предпочтениях, вебсервис ранжирует объекты по убыванию релевантности для конкретного пользователя. Рекомендательные системы помогают пользователям ориентироваться в большом числе контента, размещенного на сайте. В некоторых случаях это необходимая функциональность. К примеру, на сервисе Яндекс.Музыка по грубым оценкам размещено более 50 лет непрерывного аудио контента. Пользователь не сможет прослушать всю музыку, поэтому поиск нравящейся и при этом новой музыки пользователи осуществляют, используя советы от друзей, общественное радиовещание. В задачи рекомендательной системы в данном случае входит построение персонального аудио-потока, интересного (релевантного) пользователю, на основании его взаимодействия с проигрывателем и поведения на веб-сервисе в целом.

Результатом работы рекомендательной системы с точки зрения бизнеса является увеличение целевых показателей эффективности. Такими показателями могут служить число проданных товаров, объем продаж, доход от продаж, длительность пользовательской сессии на сайте, степень вовлеченности пользователя, лояльность пользователей к сервису. Рекомендательные системы необязательно призваны рекомендовать некоторые объекты пользователям. Для увеличения эффективности промо-акций, интернет-магазины прибегают к помощи рекомендательных систем с целью выявления наиболее заинтересованных пользователей в одном из товаров. На основании покупок пользователей, их откликов на промо-письма, поведению на витринах интернет-магазина, рекомендательная система предсказывает степень заинтересованности каждого пользователя в конкретном товаре и промо-акция проводится только для наиболее заинтересованных. В последние годы появилось множество компаний, занимающихся внедрением и развитием рекомендательных систем, наиболее известные: Gravity R&D4, RichRelevance5, Netflix6, Spotify7. Ежегодно проводится конференция «The ACM Conference Series on Recommender System» (сокращенно RecSys), на которой встречаются представители бизнесов, нуждающихся в технологии рекомендательной системы, а также исследователи в области интеллектуального анализа данных.

### 1.2 Построение рекомендательных систем

Можно выделить три основных подхода к построению рекомендательных систем:

а) на основании признаковых описаний (content-based);

б) коллаборативная фильтрация (collaborative filtering);

в) гибридный подход.

Content-based подход на основании признаковых описаний предполагает, что про пользователей и про рекомендуемые объекты известно достаточно много информации. Например, все пользователи заполняют анкету, в которой указывают свою социально демографическую информацию, интересы, и т.д. Про товары из интернет-магазина может быть известно их описание, предназначение, ценовая категория, бренд, и другие характеристики. По истории взаимодействия пользователей и объектов на сервисе можно построить обучающую выборку и свести предсказание предпочтения к хорошо изученной задаче обучения по прецедентам. На практике, использование такого похода сильно ограничено, т.к. сбор описательной информации о пользователях и объектах очень дорогостоящая процедура, которую зачастую невозможно организовать не в ущерб качеству использования сервиса, что делает рекомендательную систему неоправданно дорогой.

Гибридный подход. Несмотря на то, что алгоритмы коллаборативной фильтрации на практике показывают высокие показатели эффективности, учет дополнительной информации может сделать показатели еще выше. Одним из недостатков коллаборативной фильтрации по сравнению с методами, основанными на признаковом описании, является проблема холодного старта. Гибридный подход использует композиции алгоритмов основанных на признаковых описаниях и результатов коллаборативной фильтрации.

Коллаборативной фильтрацией называется предсказание степени предпочтения в условиях, когда рекомендательная система не обладает какойлибо описательной информацией о пользователях и объектах (либо не использует), строит прогноз исключительно на основании взаимодействия пользователей с объектами. Огромным толчком в исследовании математических моделей коллаборативной фильтрации послужил конкурс NetflixPrize. Компания Netflix занимается интернет-прокатом кинофильмов. Целью конкурса являлось улучшение качества предсказываемой оценки пользователя некоторому фильму. Набор данных конкурса содержал четверки

(дата и время, пользователь, фильм, оценка). Оценка измерялась от 1 до 5.

### 1.3 Задача коллаборативной фильтрации

Как было сказано выше, коллаборативная фильтрация — подход к предсказанию предпочтения с использованием исключительно информации о связях пользователей и объектов рекомендации. Сформулируем задачу более строго. Пусть U — множество пользователей (users), I — множество объектов (items), информация об известных предпочтениях представлена в виде набора троек: D = {(u, i, rui)}(u, i) ∈ R, где rui ∈ R — вещественная степень предпочтения объекта i ∈ I пользователем u ∈ U; R ⊆ U × I — множество пар (пользователь, объект), про которые известна степень предпочтения. Для дальнейшего удобства, введем также обозначение: R(u) = {i: (u, i) ∈ R} — множество объектов, смежных с пользователем u, аналогично: R(i) = {u: (u, i) ∈ R}. По известной информации D требуется уметь строить предсказание предпочтения ȓ ui ≈ rui для новых пар (u, i) ∈ R. Будем называть матрицей оценок матрицу R ∈ (R∪∅)|U|×|I|, строки которой соответствуют пользователям, столбцы — объектам, а элементы принимают значение rui, если (u, i) ∈ R, иначе — пропуск ∅. На задачу коллаборативной фильтрации можно смотреть как на задачу заполнения пропущенных значений в матрице. Помимо предсказания значения предпочтения, на практике могут быть интересны следующие задачи:

а) построение списка рекомендаций из объектов, на которые не

известна степень предпочтения (новые для пользователя):

RecommendedK(u)=TopK maxi ȓ ui → {(i1, rui1), (i2, rui2),..., (ik, ruik)},

б) определение степень похожести объектов и построение списков

наиболее похожих: SimilarK(i) →{(i1,sii1),(i2,sii2),...,(iK,siiK)}, где sij —

степень похожести между двумя объектами;

в) обоснование списка рекомендаций: некоторое человеко-понятное

пояснение, почему пользователю u был порекомендован объект i.

Подходы к решению задачи коллаборативной фильтрации условно можно разделить на две большие группы:

а) основанные на эвристиках (memory/heuristic-based),

б) основанные на построении модели предпочтения (model-based).

#### 1.3.1 Memory-based подход.

К этой группе методов относятся алгоритмы, выражающие пропущенные значения непосредственно через элементы матрицы оценок. Ярким примером memory-based алгоритма коллаборативной фильтрации является взвешивание предпочтения по пользователям (user-based) и по объектам (item-based).

,(1)

|  |  |  |
| --- | --- | --- |
|  |  |  |
| где |  | это средние значения предпочтений по пользователям.  , (2) |
| где |  | это средние значения предпочтений по предметам. |

В (1) и (2) представлены формулы для user-based и item-based подходов соответственно.

Мера схожести (и аналогичная для объектов) вычисляется по матрице оценок R, либо с использованием дополнительной информации о пользователях (объектах). Мера схожести является важным параметром алгоритма. Наиболее общеупотребимые простые метрики схожести — корреляция Пирсона и косинусное расстояние соответствующих строк (столбцов) матрицы оценок.

Подобные алгоритмы хороши для единоразового вычисления рекомендаций на распределенном кластере, хорошо ложатся на вычислительную архитектуру MapReduce, однако плохо подходят для оперативного обновления рекомендаций. Настройка меры схожести для задачи из конкретной предметной области является скорее искусством, нежели налаженной технологией. Одна из наиболее серьезных проблем memory-based методов — неадекватность предсказаний в условиях сильной разреженности матрицы оценок R (в смысле пропущенных значений), приводящая к невозможности подсчета метрик похожести (в случае, если множество R(u) ∩ R(u0) — пусто). Сильная разреженность матрицы оценок может быть следствием проблемы холодного старта, однако в некоторых областях она сильно разрежена всегда и не может стать достаточно плотной (например, в случае интернет-магазинов, пользователь оставляет информацию о предпочтениях в среднем двум–пяти объектам). Подобные алгоритмы не будут рассматриваться далее.

#### 1.3.2 Model-based подход.

Алгоритмы из второй группы (model-based) стремятся выбрать функцию из некоторого семейства моделей, параметризованного θ  Θ. Выбор происходит, например, путем минимизации регуляризованного эмпирического риска:

, (3)

где — функция потерь регрессии, λ — сила регуляризации, Ω(θ) — регуляризатор на множестве параметров Θ. Функция (3) отражает смысл modelbased подхода в общем виде.

## 1.4 Модель предпочтения SVD

Одна из наиболее известных моделей предпочтения — SVD, определяется следующим образом:

puTqi, θ = ({pu} u ∈ U,{qi} i ∈ I), (4)

где pu ∈ — вектор скрытых (латентных) предпочтений пользователя, qi ∈ ℝR — аналогичный вектор для объекта, R — размерность латентных векторов, будем называть ее рангом модели SVD. В формуле (4) представлена формула предсказания для рейтингов. Модель SVD обучается путем оптимизации квадратичных потерь , с квадратичной регуляризацией: Ω(θ) =

2 + 2. Название модели SVD не следует путать с сингулярным

разложением (Singular Value Decomposition). Такое название модели вошло в оборот в сообществе исследователей рекомендательных систем с момента публикации Саймоном Фанком статьи, описывающей его решение Netflix Prize. Идея подобной модели была быстро подхвачена и распространена исследователями в различных модификациях (см. обзор в статье), т.к. имела достаточно хорошую обобщающую способность (как следствие — результат в конкурсе) по сравнению с memory-based методами, популярными на тот момент. Как будет показано далее, данная модель имеет мало общего с сингулярным разложением. В своей статье, Саймон Фанк оптимизировал модель SVD методом стохастического градиентного спуска. Несколько позже был предложен более эффективный и допускающий распараллеливание по пользователям и объектам алгоритм ALS (Alternating Least Squares). Его идея заключается в том, что при фиксированных латентных векторах объектов {qi} i ∈ I, оптимизация (3) только по латентным векторам пользователей {pu} u ∈ U разбивается на независимые регуляризованные задачи наименьших квадратов по пользователям:

puTqi rᵤᵤ)2 + 2 . (5)

Для каждого пользователя u ∈ U, оптимальный латентный вектор p\*u вычисляется как решение СЛАУ с матрицей размера R×R:

P\*u qTi -1  ∙ ( ) . (6)

При фиксированных латентных векторах пользователей, оптимальные латентные вектора объектов вычисляются аналогичным образом. Алгоритм ALS заключается в циклическом пересчете латентных векторов пользователей и объектов.

1.5 Обратная связь в рекомендательных системах

Обратной связью (feedback) пользователя на некоторый объект в рекомендательных системах принято называть событие, по которому можно судить о предпочтении пользователя к объекту.

Вот несколько примеров обратной связи от пользователя:

* проставление оценки объекту по бальной шкале (количество звезд);
* нажатие на кнопку «нравится» (лайк) / «не нравится» (дизлайк); - посещение страницы с описанием объекта, переход по ссылке на объект (клик);
* посещение страницы с описанием объекта более одного раза

(заинтересованность);

* добавление в корзину / покупка объекта в случае, если это товар.

Именно по обратной связи пользователя на различные объекты, рекомендательная система формирует матрицу оценок предпочтений R, к которой затем применяются алгоритмы коллаборативной фильтрации.

Преобразование обратной связи в числовое значение предпочтения — непростая и очень важная задача в настройке рекомендательных систем. Как правило, при выборе схемы оценки предпочтения оптимизируется метрика, непосредственно связанная с ключевыми показателями эффективности (KPI) бизнеса. Техники подбора схемы оценки предпочтения выходят за рамки данной работы. По видам обратной связи, задачи моделирования предпочтения в рекомендательных системах принято разделять на два вида:

а) с явной обратной связью (explicit feedback);

б) с неявной обратной связью (implicit feedback).

Так, например, рекомендации по оценкам из пятибалльной шкалы пример задачи с явной обратной связью. Рекомендательные системы, руководствующиеся актами покупок, посещением страниц — примеры задач с неявной обратной связью. В случае неявной обратной связи имеется неопределенность в том, положительно или отрицательно влияют конкретный акт обратной связи на степень предпочтения. Покупка товара в интернетмагазине может означать достижение пользователем своей потребительской цели (положительное предпочтение), но в то же время покупатель мог после получения товара в нем разочароваться и правильно было бы засчитать негативную степень предпочтения. Очевидно, что посещения страниц пользователями веб-сервиса могут происходить при совершенно разной степени заинтересованности пользователя в контенте. Стоит отметить достаточно типичную ситуацию, когда рекомендательной системе подаются на вход исключительно положительные примеры взаимодействия пользователей и объектов. Например, веб-сервис Twitter не имеет функциональности, позволяющей пользователю выразить свое низкое предпочтение контенту, а присутствует только лишь способ «поощрить» тот или иной контент, распространив его своим подписчикам посредством функции «репост». Подобная обратная связь пользователя очень надежно (по сравнению с остальными) указывает на положительную степень предпочтения. Надежность «репостов» в сервисе Twitter подкреплена ответственностью пользователей перед своими подписчиками. В обычно выделяют три типа предположений о матрице оценок R, используемых в алгоритмах коллаборативной фильтрации:

а) все пропуски в матрице R произошли случайно (MAR, missing-atrandom);

б) все пропуски являются следствием негативного предпочтения

(AMAN, all-missingare-irrelevant);

в) пропуски в матрице R произошли не случайно (MNAR, missing-notat-random).

Под «не случайностью» выше имеется в виду вероятностное предположение о смещенности распределения предпочтения пропущенных оценок в негативную сторону. Алгоритмы коллаборативной фильтрации, использующие гипотезы AMAN и MNAR исторически принято называть «применимыми к данным с неявной обратной связью» (for implicit feedback datasets). Идея учета гипотез AMAN/MNAR породила группу алгоритмов коллаборативной фильтрации, настраивающих латентные факторы пользователей и объектов не только на известные элементы матрицы R, но и на пропущенные. Пропущенные элементы, по предположению, имеют негативную оценку предпочтения, но при этом влияют на параметры модели с меньшим весом, нежели известные.

### 1.6 Учет контекста в рекомендательных системах

Часто, для построения релевантного списка рекомендаций, рекомендательной системе нужна информация о контексте, в котором вебсервис запрашивает рекомендацию для пользователя. Вот некоторые примеры такой контекстной информации:

а) географическое положение путешественника (например, в случае

рекомендации достопримечательностей);

б) состояние корзины посетителя интернет-магазина, наличие заказа в

оформлении;

в) сезон, время, погода и другие факторы.

В простых случаях (как, например с географическим положением), достаточно фильтровать объекты рекомендаций некоторым правилом. В более сложных случаях (как, например, погода) трудно построить однозначно разумные правила фильтрации объектов.

В алгоритмах коллаборативной фильтрации с использованием контекстной информации требуется научиться предсказывать оценку предпочтения uic ≈ ruic для пользователя u  U, объекта i  I и контекста c  C, где C — заранее заданный набор контекстов. Одним из основных подходов к моделированию предпочтения с учетом контекстной информации являются факторизационные модели.

# 2 Матричная факторизация

Одним из самых популярных алгоритмов для решения проблем рекомендательных систем называется матричной факторизацией (MF). В своей простейшей форме она представлена в виде матрицы рейтингов RRm×n, которая хранит в себе информацию о оценках пользователей к предметам, где m – количество пользователей, а n – количество предметов. Применяя этот алгоритм, можно разложить матрицу R на произведение 2 других матриц

U  Rm×k и Pn×k. Данная операция продемонстрирована на рисунке 2 для наглядности.

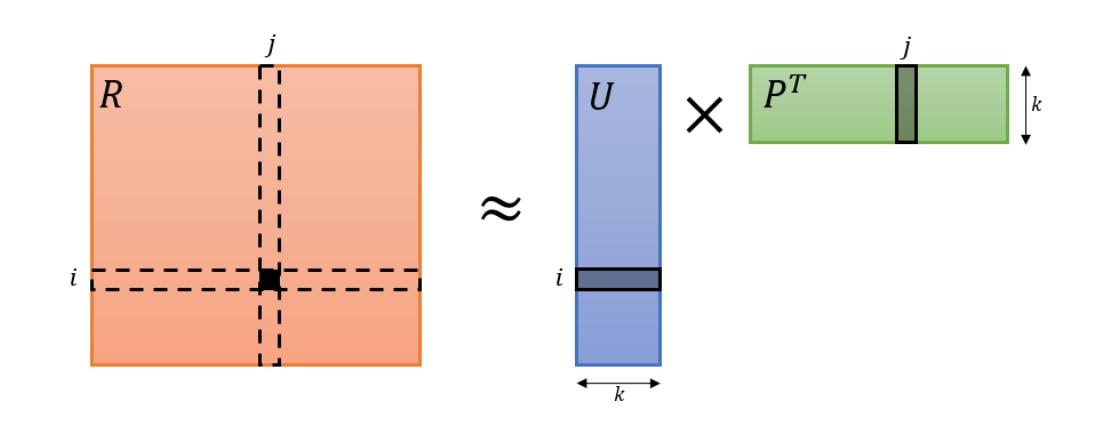


Рисунок 2 – Факторизация матрицы

Обратите внимание, что этот алгоритм вводит новую величину, k, которая является размерностью как в матрице U так и в матрице P. Это значение называют рангом факторизации. Формально, каждый рейтинг Rij для пользователя i к предмету j можно вычислить как скалярное произведение векторов Ui ˙ Pj , размерность которых равна k.

Итак, предполагая, что R представляет собой матрицу пользователей, которые оценивают фильмы, то довольно очевидно, что каждый фильм может быть оценен по совокупности критериев, каждый из которых равен некоторой величине. Каждому пользователю нравятся либо не нравятся фильмы с определенными критериями(например, не нравится жанр комедии). То есть чтобы посоветовать пользователю подходящий фильм необходимо найти такой, критерии которого будут схожи с критериями пользователя. Проблема в том, как эффективно формировать эти категории. Черт, он может даже смотря некоторые сходства с некоторыми актерами кино, режиссерами, язык, место съемок и больше, а количество возможных функций для создания огромны.

Проблема в том, что мы не знаем какое количество критериев k на самом деле.

В итоге мы имеем попарное взаимодействие между двумя сущностями (пользователь и фильм), что позволяет нам поставить некий рейтинг в матрицу R на основе этих взаимодействий, которая довольно разряжена (из-за большого разнообразия фильмов, так как пользователи могут оценить лишь малую часть из них). Теоритически можно разделить фильмы по категориям используя как критерии: жанр, язык, актеры и так далее. Просто проблема в том, что мы не можем качественно оценить влияние каждого критерия на итоговую оценку фильма. Интуитивно, можно представить пользователям (или фильмы) по их сходству (или соединение) в категории/актеры/языков или каких-либо других факторов, и, наконец, попытаться сгруппировать их на основе этого представления. Проблема в том, что довольно трудно дать количественную оценку влияния каждого такого фактора на исходные рейтинги. Кроме того, могут существовать факторы, которые присущи рейтингового процесса, но не упустил из нашего нового представительства. Тем более возможно существование таких критериев, которые влияют на рейтинг, но не задействованные в наших вычислениях.

Можно заметить, что MF - это задача оптимизации, который направлена на максимальное приближение произведения матриц U и P к исходной матрице рейтингов R. В алгебраическом виде задача представлена в виде уравнения (6):

. (6)

То есть минимизировав уравнение (6), мы минимизируем квадратичную ошибку, что позволит получить приемлемые матрицы U и P. Значение является параметром регуляризации модели. Регуляризация модели — это способ наложить штраф к целевой функции за сложность модели. С байесовской точки зрения — это способ учесть некоторую априорную информацию о распределении параметров модели. Важным свойством является то, что регуляризация помогает избежать переобучения модели.

Это проблема оптимизации удобно решается с помощью методов машинного обучения. Заметим, что эта функция содержит два параметра: k и . Помимо осуществления минимизации значения функции (6), важно качественно подобрать параметры K и так, чтобы они давали оптимальные результаты. Далее рассмотрим основные подходы минимизации функции

##### 2.1 Градиентный спуск

Градиентный спуск-это алгоритм оптимизации, который широко используется в области машинного обучения. В принципе, это простой итеративный алгоритм оптимизации, который предполагает наличие функции и произвольных начальных значений для переменных оптимизации.

Градиентного спуск на каждой итерации повторно вычисляет градиент функции относительно параметров оптимизации и затем обновляет их, используя антиградиент функции. Спуск происходит до тех пор, пока значение градиента с текущими параметрами оптимизации не будет равно 0. Однако, заметим, что оптимизация функции через алгоритм градиентного спуска гарантирует лишь сходимость к локальным минимумам.

Метод градиентного спуска хорошо применим к задаче оптимизации матричной факторизации(MF). Однако, этот алгоритм не дает лучших результатов при оптимизации(MF), если размерность исходной матрицы клиентов очень велика, так как там необходимо оптимизировать параметров. Так как в реальных задачах это часто случается, то необходим другой механизм, который будет склонен к распараллеливанию.

##### 2.2 Метод чередующихся наименьших квадратов

(7)

В уравнении (7) фактическая функция (6) была выражена через сумму. Тот факт, что переменные U и V неизвестные, позволяет сделать вывод, что эта функция не является выпуклой.

Также еще один интересный факт заключается в том, что если мы зафиксируем P и оптимизируем U, то проблема просто сводится к задаче линейной регрессии.

Алгоритм чередующихся наименьших квадратов представляет собой двухэтапный итеративный процесс оптимизации. В каждой итерации сначала он фиксирует P и решает задачу для U, затем, он фиксирует U и решает для P. Чередование двух шагов происходит определенное количество итераций или до сходимости функции. По аналогии с градиентным спуском, этот алгоритм гарантированно сходится только к локальным минимумам, и, в конечном счете, зависит от начальных значений U или P.

, (8)

. (9)

В (8) и (9) представлены функции для нахождения латентного вектора для пользователя i и предмета j соответственно.

Также стоит отметить, так как все вектора пользователей не зависят друг от друга( с предметами аналогично), то эти моменты можно распараллелить. В дальнейшей работе будем использовать алгоритм чередующихся наименьших квадратов.

# 3 Практическая реализация рекомендательной системы с неявной обратной связью через матричную факторизацию.

Перед написанием кода для рекомендательной системы, обговорим некоторые моменты, а именно как работает ALS метод и почему именно он был реализован как единственным метод из аналогичных методов в библиотеке Spark. Прежде уже говорилось, что для коллаборативной фильтрации не требуется какой-либо информации о пользователях или предметах. Следовательно, может задаться вопрос как пользователи и элементы связаны друг с другом?

Оказывается, можно найти связь между пользователями и предметами, если использовать факторизацию матрицы. Часто, факторизация матрицы применяется в области сокращения размерности, где необходимо сократить количество исходных данных, сохранив при этом необходимую информацию.

Это можно сделать через сингулярное разложение матрицы(SVD).

По сути, можно взять большую матрицу взаимодействия пользователей с предметами и выявить латентные (или скрытые) особенности, которые позволят разложить эту матрицу на две другие матрицы, которые на порядок меньше, и в которых находятся латентные вектора пользователей и предметов соответственно. Это именно то, что ALS пытается сделать через факторизацию матрицы.

Рассмотрим рисунок 3, который расположен ниже, предположим, имеется оригинальная матрица оценок RM×N, где M-количество пользователей и Nколичество предметов. Эта матрица является довольно разряженной, поскольку большинство пользователей взаимодействуют только с несколькими элементами. Можно разложить эту матрицу через произведение двух других матриц более низкого порядка: одна с размерами M×K, которая будет хранить латентные вектора признаков для всех наших пользователей (U), а вторая с размерами K×N, в которой будут находится латентные вектора признаков для каждого предмета (V). Перемножив эти две матрицы получим матрицу, которая будет равна исходной матрице R. Теперь получились две матрицы, которые являются плотными, и в которых хранятся латентные вектора признаков для каждого из предметов и пользователей.

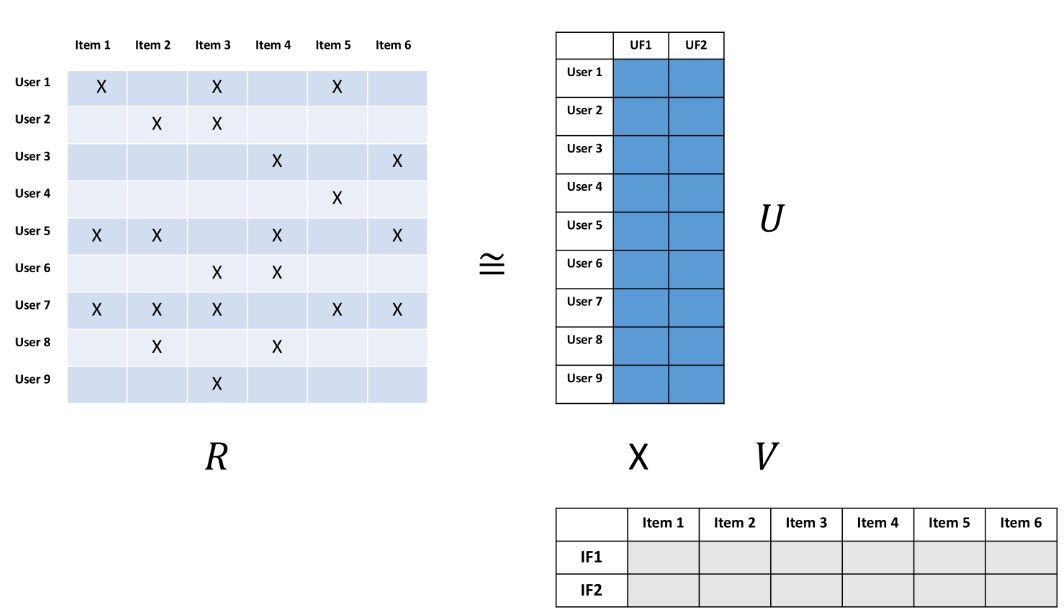


Рисунок 3 – Факторизация матрицы

На рисунке 3 изображено разложение матрицы на произведение двух других более мелких матриц. Для того, чтобы найти значения матрицы U и V, можно либо использовать SVD (что потребует инвертирование потенциально очень большие матрицы, а это будет слишком дорого стоить в вычислительном отношении) для более точного решения задачи факторизации, или применить алгоритм ALS и получить приближенное решение. В случае с ALS, так как и множество пользователей U и множество предметов V являются независимыми, то есть надо найти решение лишь для одного вектора признаков в один момент времени, что означает, что он может быть запущен параллельно! (В этом просматривается большое преимущество и, возможно, именно поэтому этот метод был выбран для библиотеки Spark). Чтобы использовать метод ALS, можно случайным образом инициализировать U и найти подходящее решение для V. Затем нужно вернуться и найти решение для U, используя решение для

V. Эти действия выполняются до тех пор, пока мы не получим приближение к матрице R, которое будет наилучшим.

После того как были получены подходящие матрицы U и V, для того чтобы предсказать оценку пользователя u к объекту i можно просто взять скалярное произведение u-ой строки матрицы U и i-ого столбца матрицы V, и что дает рейтинг пользователя u к этому предмету i, даже если ранее пользователь u не взаимодействовал с предметом i. Эта базовая методология была принята для рекомендательных систем с неявной обратной связью в статье «[Collaborative Filtering for Implicit Feedback Datasets»](http://yifanhu.net/PUB/cf.pdf). Далее рассмотрим эту методологию на конкретном примере.

### 3.1 Обрабатываемые данные

В качестве входных данных рассмотрим взаимодействие пользователей и товаров в интернет магазине. На рисунке 4 изображена структура файл данных.

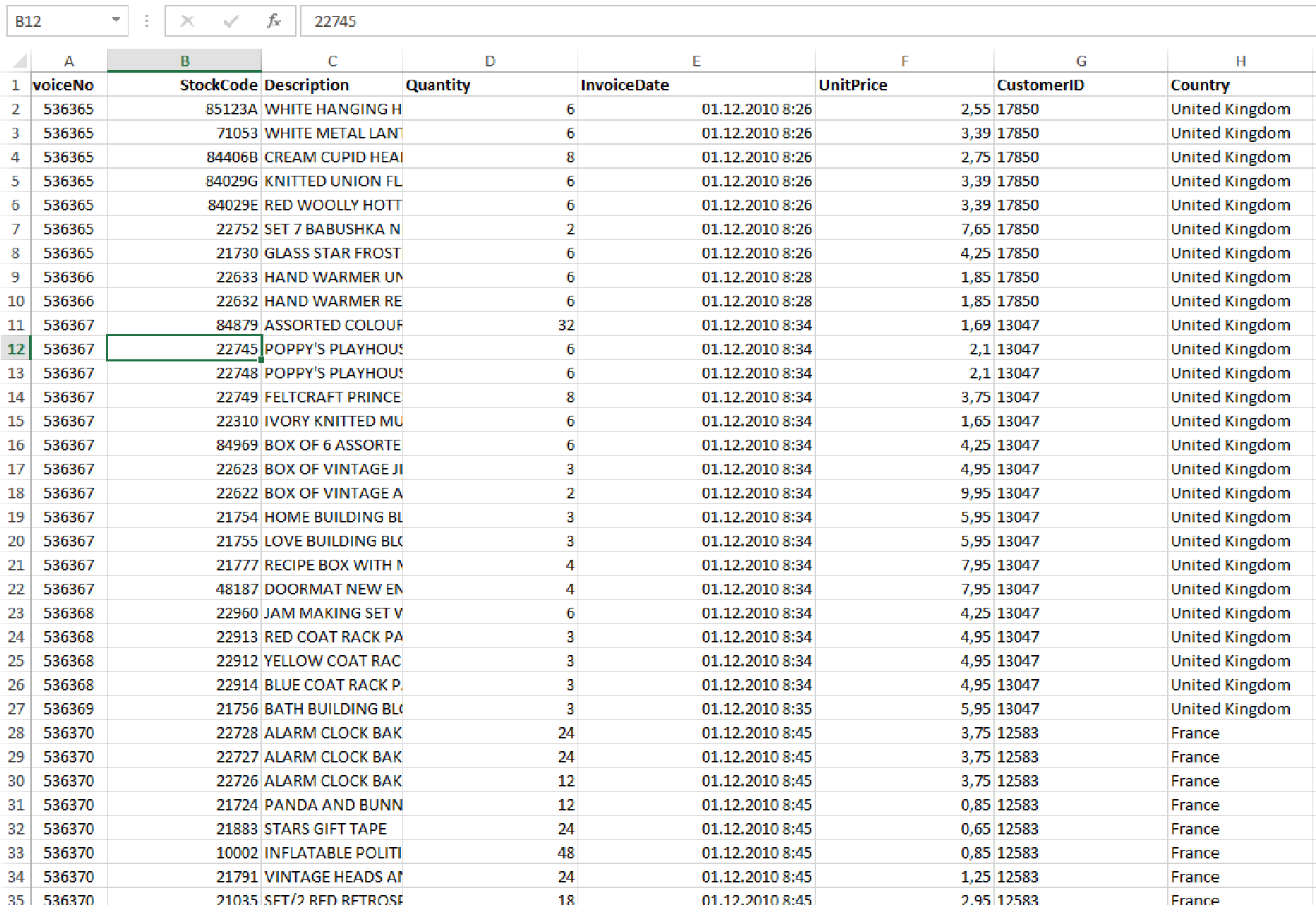


Рисунок 4 – Файл с транзакциями пользователей

Рассмотрим все сделки для каждого клиента и представить их в формате, который будет пригоден для ALS. Это означает, что нам необходимо представить данные в виде матрицы так, чтобы все ID клиентов были заголовками в строках матрицы, а все ID элементов были заголовками в столбцах матрицы. Такая матрица заполняется количеством покупок товаров каждым из клиентов.

Перед написанием кода, загрузим некоторые библиотеки, которые помогут нам с предварительной обработкой:

а) pandas;

б) scipy;

в) numpy

Набор данных включает в себя номер счета-фактуры для различных покупок, наряду с StockCode (или ID товара), описание товара, количество покупок, дата покупки, цена товара, идентификатор заказчика, и страна происхождения для заказчика. Чтобы избежать ошибок, необходимо проверить все ли поля, которые планируется использовать, заполнены.

Большинство колонн не имеют пропущенных значений, но ID клиента отсутствует в нескольких строках. Если ID клиента отсутствует, мы неизвестно, кто купил товар. Необходимо отбросить такие строки из данных. Будем использовать использовать функцию pandas.isnull, чтобы проверить строки с недостающими данными и оставить только те строки, которые имеют идентификатор клиента.

Прежде чем сделать какую-либо матрицу оценок, было бы неплохо иметь таблицу, в которой хранятся ID товара и его описание. Это позволит составить соответствие между StockCode и его описанием. Затем выполним следующие действия, которые подготовят данные к использованию:

а) сгруппировать количество закупок вместе по StockCode и ID товара;

б) менять любые суммы, которые равны нулю на один (это может произойти, если товары были возвращены, так будет обозначено, что пользователь фактически приобретал товар, чтобы отличать отсутствие взаимодействия между пользователем и товаром с

возвратом товара);

в) создать наш рейтинг разреженные матрицы.

Этот последний шаг является особенно важным, так как позволит бороться с возможными проблемами памятью компьютера. Очевидно, что в этом случае, матрица будет содержать тысячи товаров и тысячи пользователей с рейтингами для этих товаров. Это большая матрица, поэтому мы можем сэкономить много памяти на хранении разреженной матрицы, если будем хранить только значения элементов, которые не равны нулю.

Код ниже (рисунок 5) выполнит финальные шаги предварительной обработки, необходимой для получения матрицы подходящего вида.

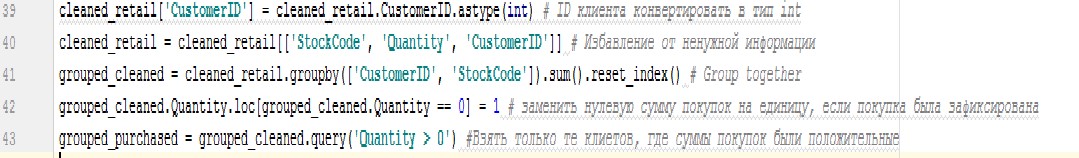


Рисунок 5 – Предварительная обработка

Вид результирующей матрицы сгруппированных покупок (рисунок 6).

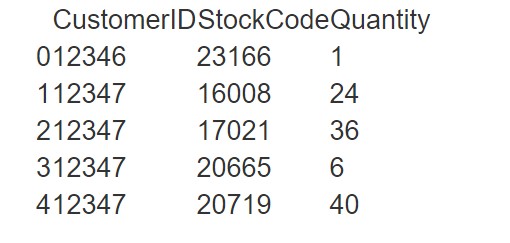
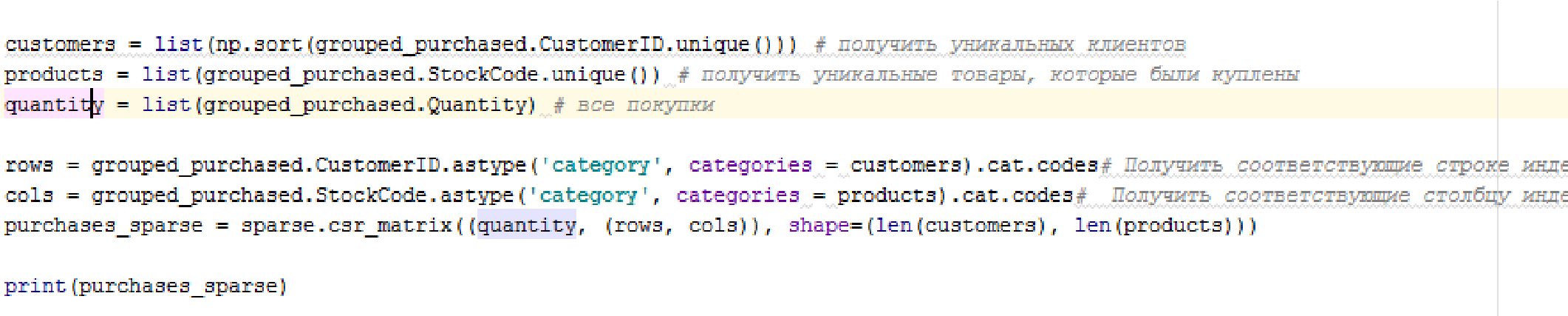


Рисунок 6 – Полученные отфильтрованные данные

Вместо того, чтобы представлять явный рейтинг (например, оценку от 1 до 5), количеством закупок товара можно представлять «уверенность» в плане того, как сильно этот товар подходит данного пользователю. Предметы с большим числом покупок клиентом могут иметь большой вес в матрице рейтингов покупок.

Так как матрица сильно разряженная, то стоит позаботится о ее структуре, чтобы не расходовать память в пустую. Создание разреженной матрицы рейтингов пользователей и элементов, используя код ниже (рисунок 7).

Рисунок 7 - Создание разряженной матрицы

В итоге получилось получилась 4338 клиентов и 3664 товаров. При взаимодействии между пользователями и товарами, пользователями всего было куплено 266723 товаров. В условиях разреженности матрицы взаимодействия равной 98.3% мы можем применить коллаборативную фильтрацию. Для работы алгоритмов коллаборативной фильтрации необходимо, чтобы разреженность была менее 99,5% иначе можно не получить хороших результатов. В данном случае разряженность далека до этого, поэтому использование коллаборативной фильтрации должно вернуть достойные результаты.

3.2 Создание обучающего и тестового множеств для РС.

Как правило, в приложениях машинного обучения, производится проверка того, что является ли модель, которая уже считается натренированной на обучающей выборке, хорошо работающей на новых данных, которые она еще не видела на этапе обучения. Это можно сделать путем создания тестового набора полностью отделенного от обучающего набора. Обычно это довольно просто сделать: достаточно взять случайную выборку обучающих примеров из нашей матрицы и отделить ее от обучающего набора (рисунок 9). Это обычно выглядит так:

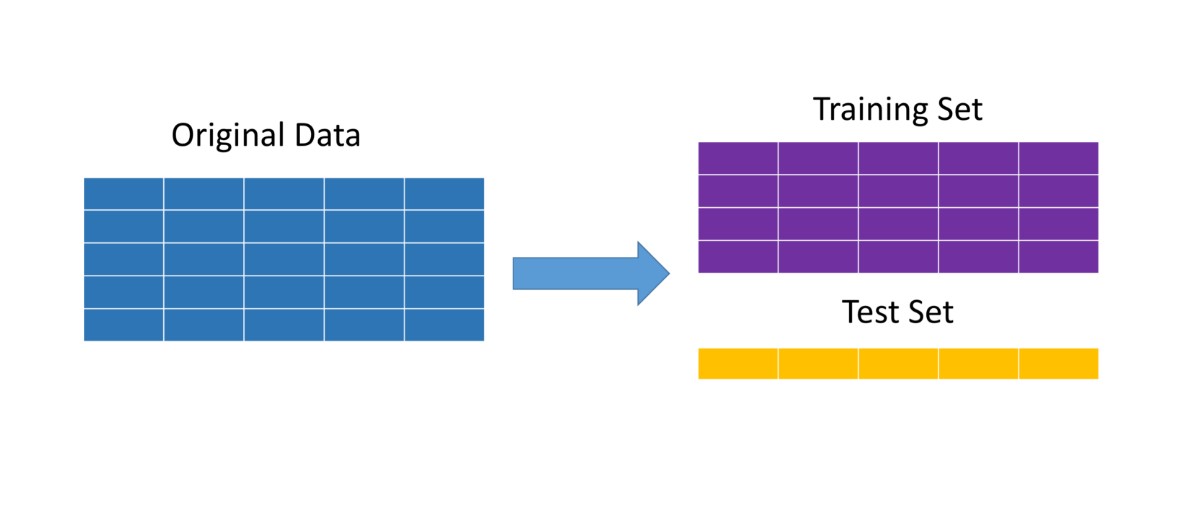


Рисунок 9 – Показательное разбиение данных на 2 выборки

С коллаборативной фильтрацией, это не будет работать, потому что в ней нужны все взаимодействия пользователей и элементов, чтобы найти правильную факторизации матрицы. Лучший способ скрыть определенный процент взаимодействий пользователей и предметов в модели на этапе тренировки. Стоит отметить, что выбирать скрываемые взаимодействия можно случайно. Затем, проводится проверка на стадии тестирования, выясняется какие из многих товаров, которые были рекомендованы пользователю, на самом деле в конечном итоге были куплены. Также можно проверить рекомендательную систему с использованием данных из временных рядов, где все данные до определенного момента времени используется для тренировок, а затем после определенного периода времени используется для тестирования. Как можно разбить исходные данные на две обучающие выборки данных проиллюстрируем на рисунке 10:

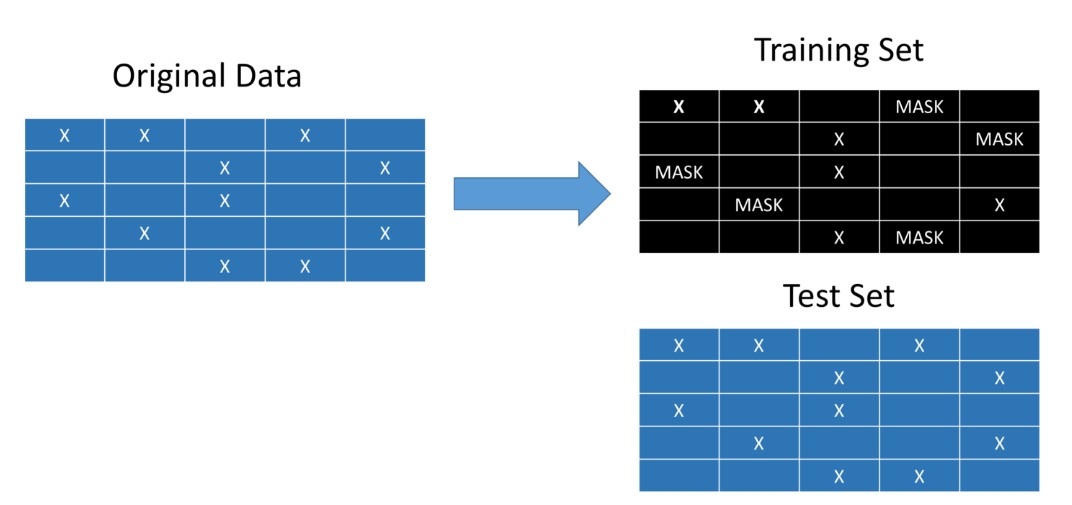


Рисунок 10 – Визуализация наложения маски

В данном случае тестовый набор - это точная копия исходных данных. Обучающий набор, однако, будет маскировать случайный процент взаимодействий пользователей с объектами, и действовать, как если бы пользователь не купил эти товары (то есть значение матрицы относительно этих определенных пользователей и объектов будет равно нулю). После обучения модели, проверим её на тестовом наборе данных, и выясним, действительно ли товары, которые были рекомендованы пользователю оказались им куплены. Если пользователи в конечном итоге часто приобретали товары, которые им рекомендовала система, то можно сделать вывод: система, кажется, работоспособной.

В качестве дополнительной проверки рекомендательной системы, можно сравнить получившуюся систему с наиболее популярными товарами. Этот метод тестирования не обязательно дает достоверный результат, потому что это зависит от того, как будет использоваться реализуемая система рекомендаций. Однако, это практический способ тестирования производительности, поэтому также будем использовать его для данного примера.

Определившись с способом выделения тренировочной выборки и тестовой выборки, создадим функцию, которая выполнит это (рисунок 10).

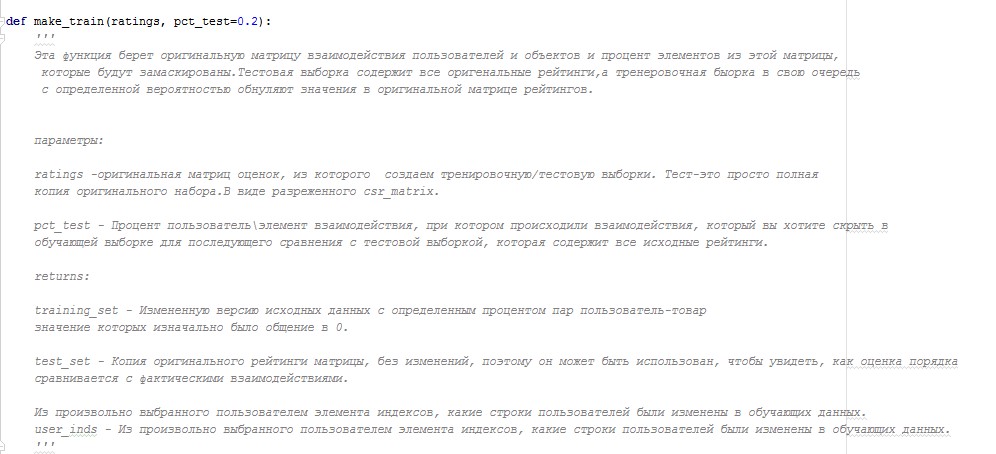


Рисунок 11 – Функция генерации тренировочной выборки

Эта функция вернет обучающий набор, набор тестов, который был бинаризован, 0/1 за купленные/не купленные, и список пользователей, которые взаимодействовали по крайней мере с одним скрытым товаром. Тестирование производительности рекомендательной системы будет произведено именно на этих пользователях. В данном случае было замаскировано 20% взаимодействий пользователей с товарами.

Приведя исходные данные к нужному виду перейдем к реализации алгоритма чередующихся наименьших квадратов.

##### 3.3 Реализация ALS для рекомендательной системы с неявной обратной связью на языке Python.

Имея тренировочные и тестовые наборы данных, можно переходить к реализации алгоритма.

Вспомним, что имеем матрицу оценок, которая сильно разрежена. Получим из нее матрицу уверенности. То есть тем самым в случае соответствующих действий пользователя закрепляем его уверенность относительно некоторых объектов.

, (10)

где – это уверенность в том, что пользователю u подходит предмет i. Переменная α представляет собой линейное масштабирование рейтинга предпочтений (в нашем случае количество покупок) и переменная rui является значением из нашей оригинальной матрицы закупок. Формула (10) описывает получение матрицы уверенности для всех пар клиент – предмет.

После взятия производной из уравнения 3 в статье [1], мы можем минимизировать получившуюся функцию для пользователей U:

, (11)

Авторы из статьи [1] утверждают, что можно увеличить производительность этого вычисления, если его немного изменить с помощью линейной алгебры:

, (12)

Стоит Обратить внимание, что теперь можно предварительно рассчитать часть YTY для того, чтобы не вычислять его заново для каждого пользователя u.

Также аналогичное уравнение можно получить и для предметов:

(13)

Эти два уравнения необходимо чередовать до тех пор, пока они не сойдутся. Также в этой модели есть коэффициент регуляризации λ, который может помочь предотвратить переобучение на этапе обучения, а также, также здесь есть матрица предпочтений p, которая заполнена нулями и единицами, причем единица стоит лишь в том случае, если между пользователем и предметом было взаимодействие.

Теперь представим эту концепцию в виде кода (рисунок 12).

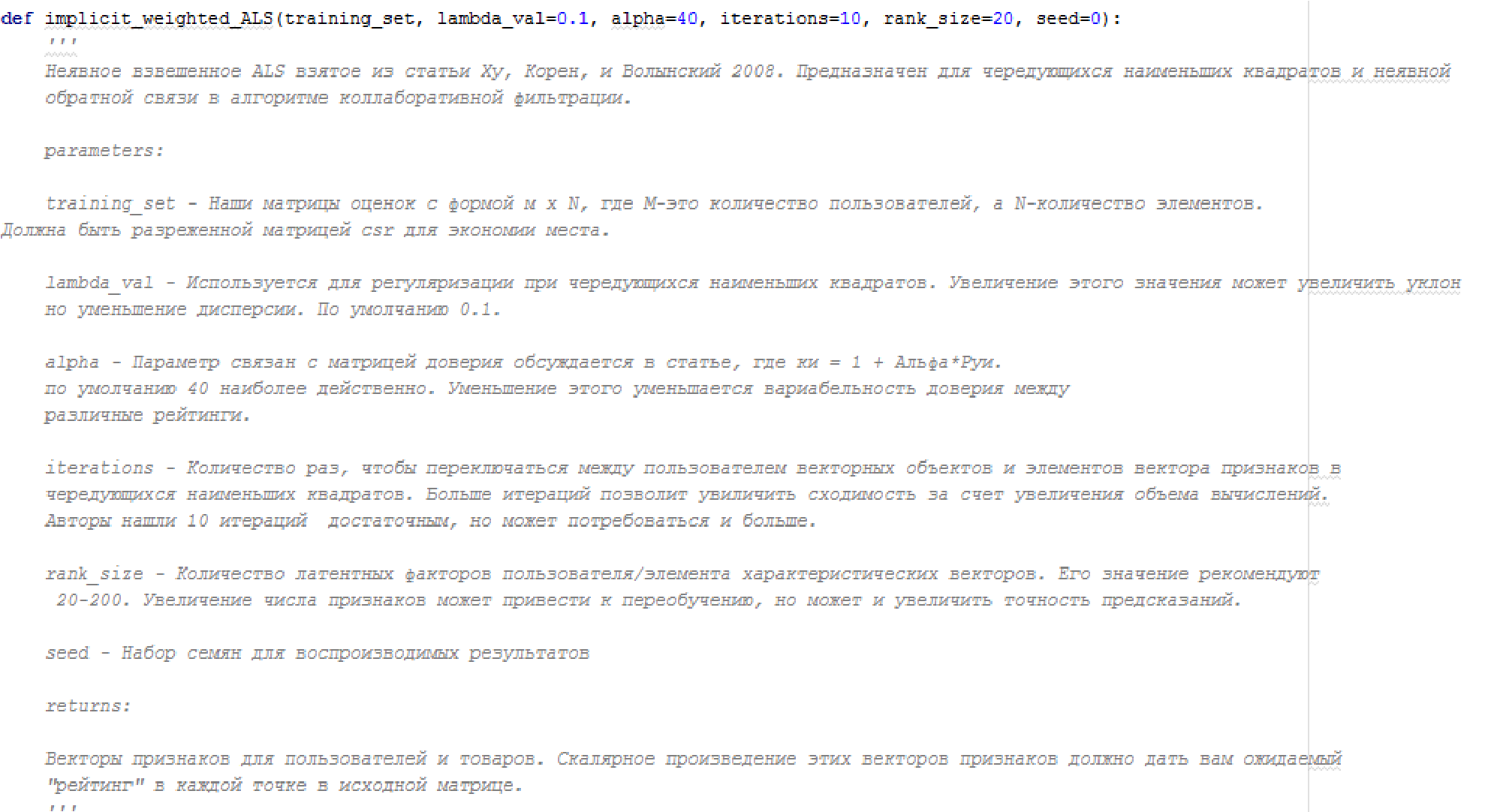


Рисунок 12 – Реализация ALS

Попробуем просто запустить одну итерацию кода, чтобы увидеть, как это работает (это довольно медленно прямо сейчас!). Выберем 20 скрытых факторов вместе с альфой 15 и регуляризацией 0.1. Это заняло около 70 секунд.

Теперь после исполнения кода можно исследовать рейтинги для конкретного пользователя, используя скалярное произведение между пользовательским вектором из матрицы U и векторов товаров матрицы V.

При данной реализации была выполнена только одна итерация алгоритма ALS, так как он работает слишком медленно. Конечно, следовало выполнить как минимум десять итераций, так чтобы U и V сходились. Так как эта реализация слишком ресурсно-затратна, следует двигаться дальше.

##### 3.4 Ускоренный ALS

Этот код в необработанном виде работает слишком медленно. Мы должны сделать много циклов, следует помнить, что этот алгоритм можно легко запустить параллельно, то есть вычисление латентного вектора пользователя u может быть вычислен независимо от других пользователей. На языке Python а библиотеке implicit присутствует такая реализация. Это более чем в 1000 раз быстрее, чем обычная реализация на чистой версии Python.Эта библиотека легко может быть установлена, если у вас есть pip, то выполните команды pip install implicit.

Эта версия кода не имеет параметра α и подразумевается, что α должна быть внедрена в матрицу оценок. Это нужно сделать, прежде чем использовать её в качестве входных данных. Здесь можно экспериментировать с количеством факторов, альфой и регуляризацией, чтобы найти более подходящие параметры для данной модели. Тип входной матрицы должен быть double, чтобы функция

ALS работала корректно (рисунок 13).

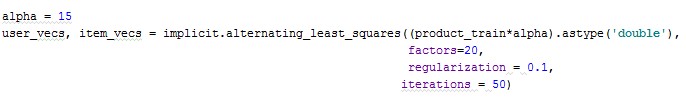


Рисунок 13 – Использование ускоренного ALS

Теперь код выполняется намного быстрее, но нам надо убедится, что система выдает результаты на приемлемом уровне.

### 3.5 Оценивание рекомендательной системы

При выделении обучающей выборки было скрыто 20% всех покупок. Это позволило оценить эффективность рекомендательной системы. По сути, видно, что предсказанные рекомендации для каждого пользователя совпадают с теми, которые они в конечном итоге купили. Часто используемой метрикой для такого рода проблемы является рабочая характеристика приёмника (или ROC) кривой. Большая площадь под кривой означает, что система рекомендует товары, которые в конечном итоге были приобретены. Как правило, эта метрика используется в более типичных проблемах бинарной классификации, чтобы определить, насколько хорошо модель, может прогнозировать положительный или отрицательный пример. Он также будет хорошо работать для наших целей ранжирования рекомендаций.

Чтобы это сделать, необходимо написать функцию, которая сможет вычислять среднюю площадь под Кривой (AUC) для любого пользователя, у которого взаимодействие хотя бы с одним товаром скрыто. В качестве ориентира вычислим среднюю AUC, то есть рекомендуемая выборка из самых популярных товаров. Популярность обычно является трудно победимым фактором в большинстве рекомендательных систем, поэтому она даёт хорошее сравнение.

Для этих целей создадим простую функцию, которая может вычислить наш AUC (рисунок 14). Пакет scikit-learn имеет подходящую функцию, которую надо немного изменить.

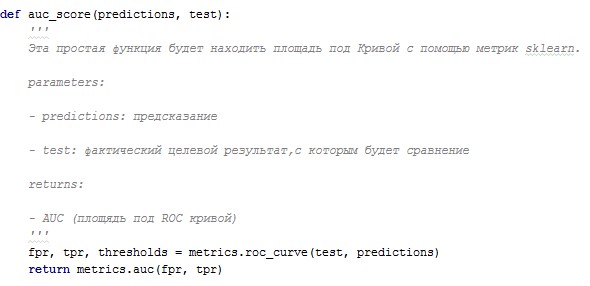


Рисунок 14 – Функция нахождения площади под кривой

Теперь используем эту вспомогательную функцию внутри второй функции, которая будет вычислять значение AUC для каждого пользователя в нашем обучающем наборе, которые имели по крайней мере скрытым по крайней мере одно взаимодействие с товарами (рисунок 15). Следует также рассчитать AUC самых популярных товаров для наших пользователей, чтобы затем сравнить с результатом рекомендаций системы.

Рисунок 15 – Функция вычисления AUC

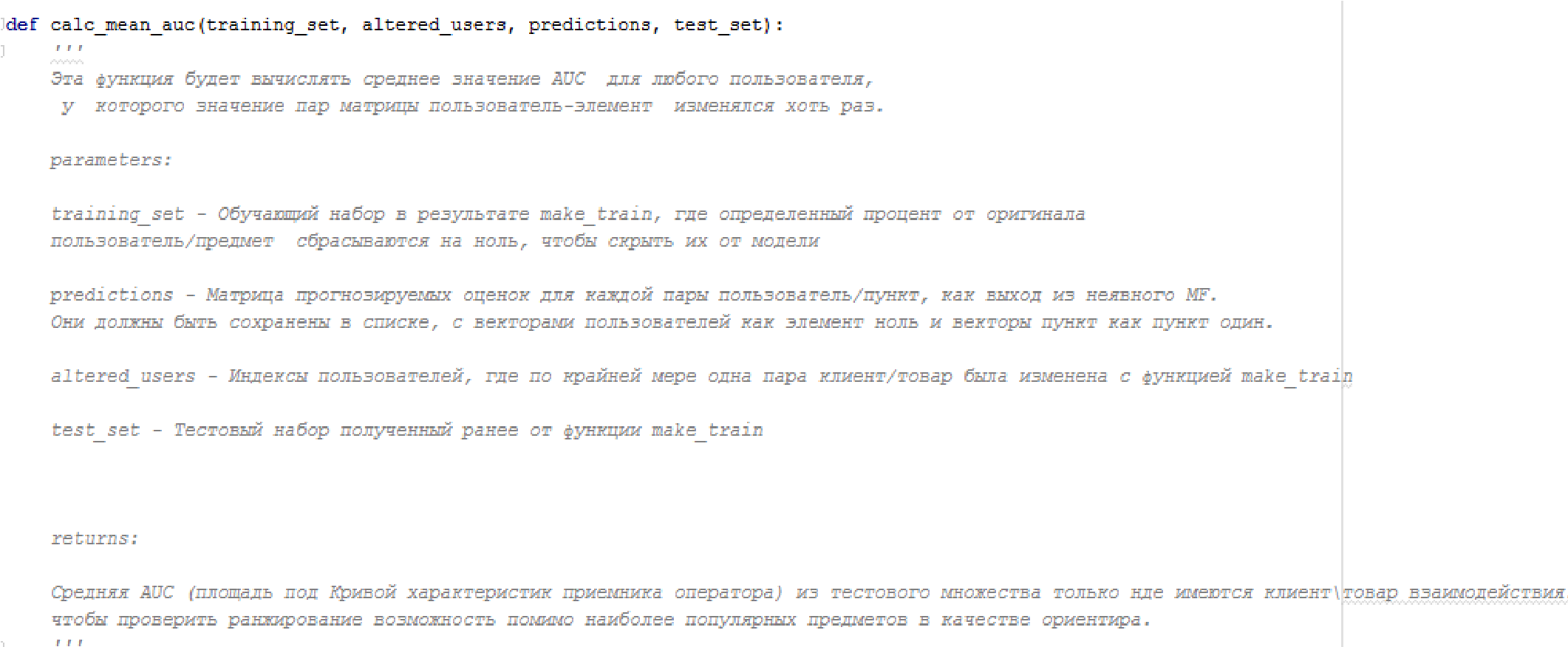
Чтобы использовать эту функцию, необходимо преобразовать выходные данные из функции ALS в csr\_matrix формат и транспонировать вектора товаров. Получение векторов пользователей и элементов в уже нужном формате изображен на рисунке 16.



Рисунок 16 – Получение векторов пользователей и элементов

Здесь видно, что система рекомендаций превысила точность наиболее популярных товаров. У система рекомендаций среднее значение AUC в размере 0,87, в то время как наиболее популярные товары получили более низкий AUC из 0.814. Также не стоит забывать, что это параметризированная модель и при определенных параметрах можно увеличить AUC.

##### 3.6 Пример выдачи рекомендаций

Теперь получена рекомендательная система, что она имеет более точное предсказание чем рекомендация самых популярных товаров. AUC равный 0,87 означает, что система рекомендует пользователю такие товары, которые он бы на самом деле купил при обучении на тестовом множестве намного чаще, чем те, с которыми он никогда не взаимодействовал. Чтобы увидеть на примере, как это работает, рассмотрим рекомендации для конкретного пользователя и субъективно решим, если ли в этих рекомендациях какой-то смысл.

Для начала необходимо реализовать функцию, которая будет возвращать для пользователя товары, с которыми он уже взаимодействовал. Для этих целей создаем массивы клиентов и товаров, которые были сделали заранее, что показано на рисунке 17.



Рисунок 17 – Инициализация ID клиентов и товаров

Теперь можно создать функцию, которая будет возвращать список товаров с которыми пользователь уже взаимодействовал (рисунок 18).

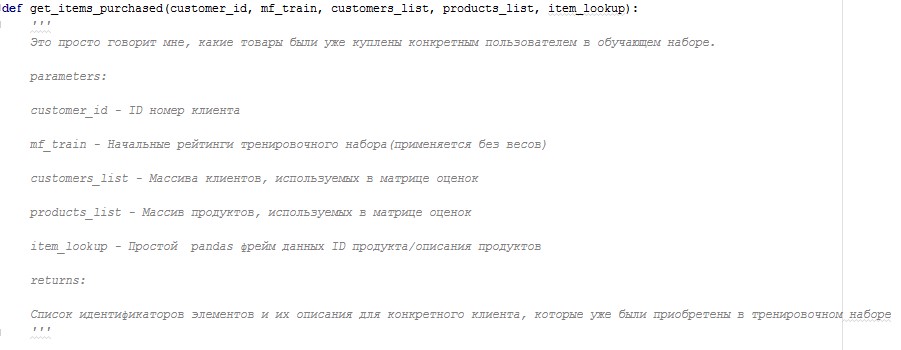


Рисунок 18 – Получение взаимодействующих с клиентом товаров

Необходимо искать их по ID клиента (рисунок 19). Глядя на список клиентов:



Рисунок 19 – Список ID клиентов

Мы видим, что первый в списке клиентов имеет ID равный 12346.

Рассмотрим его покупки из обучающего набора (рисунок 20).



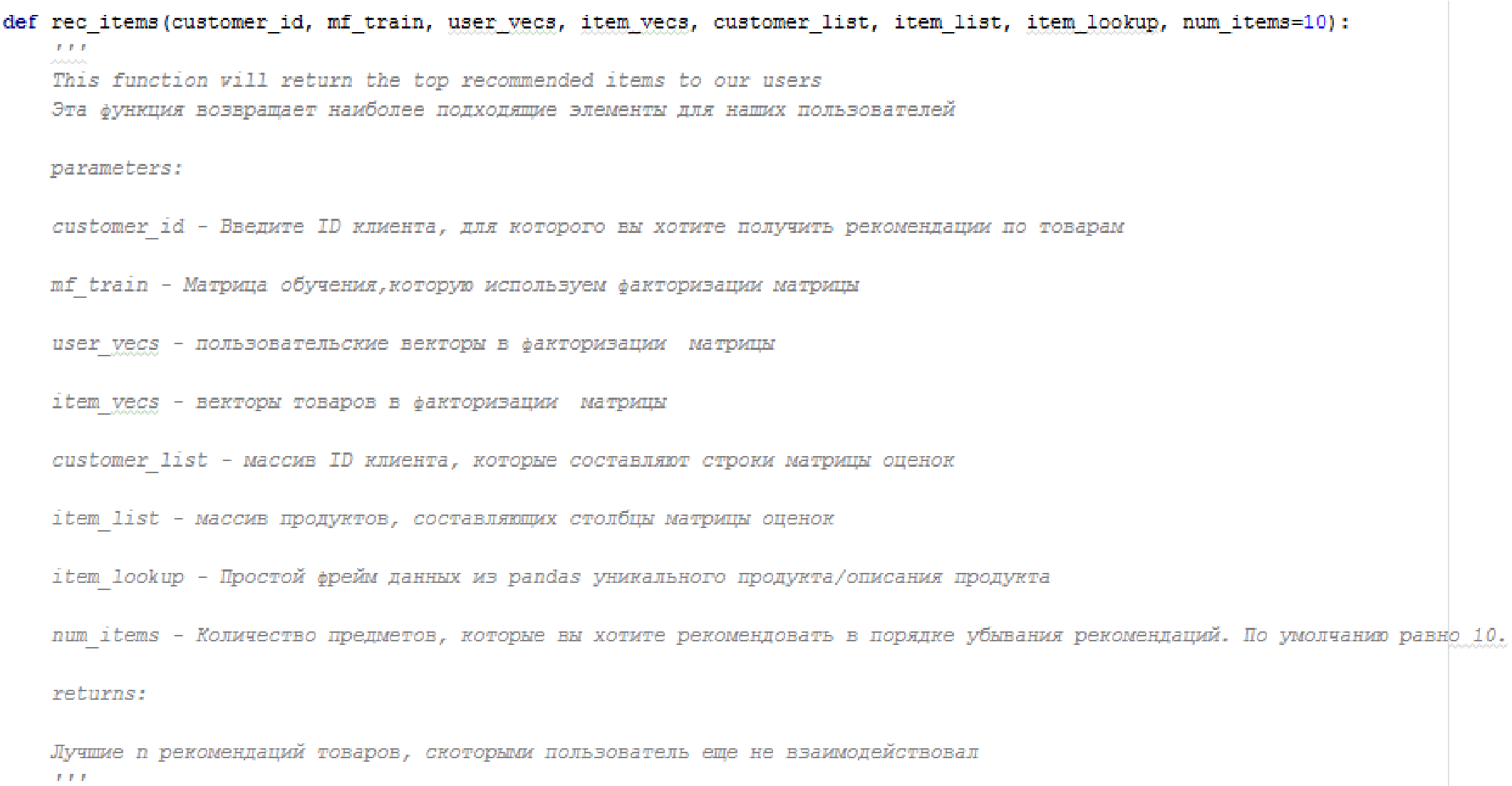
Рисунок 20 – Получение товаров, которые взаимодействовали с текущим клиентом

Мы видим, что клиент приобрел MEDIUM CERAMIC TOP STORAGE JAR.

Теперь зная о покупках пользователя, реализуем функцию, которая на основе предыдущих предпочтений представит пользователю рекомендации. Для написания такой функции импортируем также MinMaxScaler из пакета scikitlearn (рисунок 21), что упростит нам ее описание.

Рисунок 21 – Получение рекомендаций

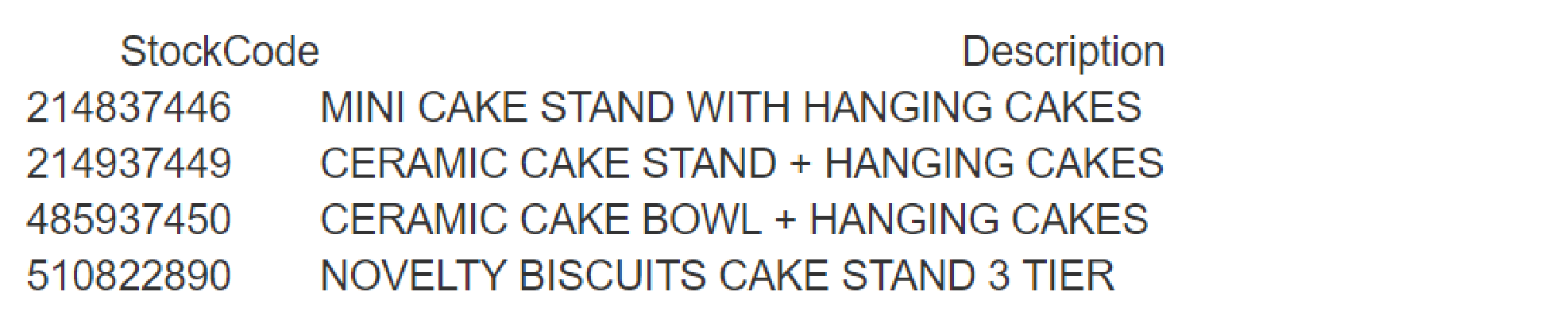
По сути, эта функция будет возвращать N самых больших скалярных произведений между конкретным пользователем и всеми векторами товаров. Предметы, которые уже были приобрели конкретным пользователем больше ему не рекомендуются. Будем возвращать по умолчанию 10 товаров и рассмотрим, что рекомендательная система порекомендовала для конкретного клиента с ID 123456 (рисунок 22).



.

Рисунок 22 – Полученные рекомендации

Эти рекомендации кажутся довольно хорошими. Помните, что система рекомендации не имеет никакого реального понимания того, что из себя представляет керамическая фляга. Все, что она знает, так это историю покупок. Система установила, что люди, купив банку средних размеров также могут купить баночки разного размера. Я лично был поражен тем, насколько хорошо такая система подходит к подобной модели шопинга. Теперь рассмотрим другого пользователя с ID = 12353, который не сделал большого количества покупок (рисунок 23).

Рисунок 23 – Покупки пользователя с ID = 12353

По покупкам этого пользователя кажется, что он печет пирожки. Рассмотрим какие товары рекомендательная система предложит этому человеку (рисунок 24).



Рисунок 24 – Полученные рекомендации

Система выдала выборку на тему тортов вместе с керамическими изделиями. Опять же, эти рекомендации кажутся очень впечатляющими, так как система не понимает содержание рекомендаций. Теперь рассмотрим рекомендации для ID = 12361 (рисунок 25).

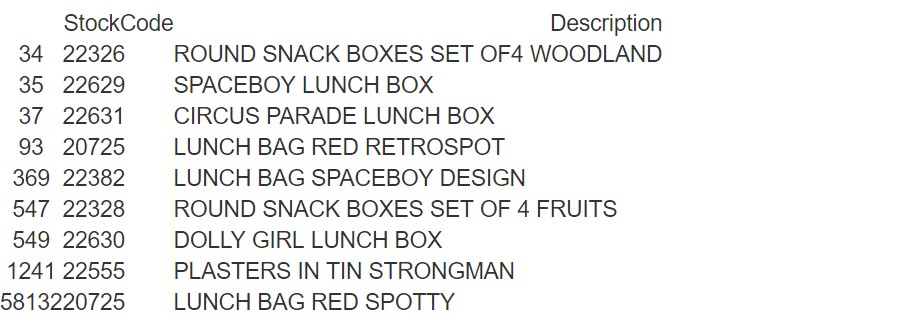


Рисунок 25 – Приобретенные товары текущим клиентом

Этот клиент покупал продукты, пригодные для обеда. Теперь давайте рассмотрим, что выдала рекомендательная система (Рисунок 26).

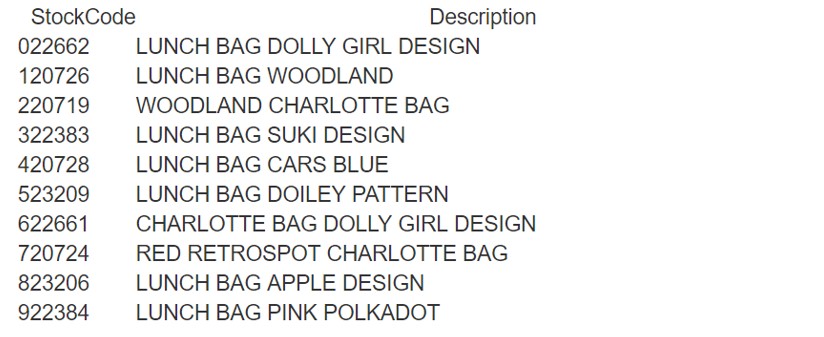


Рисунок 26 – Полученные рекомендации для данного пользователя

Опять же, рекомендательная система удивительно неплохо справилась.

Много товаров, связанных с едой, представлено в этом списке рекомендаций.

# ЗАКЛЮЧЕНИЕ

В этой работе были сделаны выводы о том, какие преимущества и недостатки существуют у алгоритмов коллаборативной фильтрации, было произведено ознакомление с их устройством и основными подходами при решении задачи рекомендаций с неявной обратной связью. Также изучена и воспроизведена на практике разработка рекомендательной системы с неявной обратной связью и получение рекомендаций. Также было рассмотрено как работает рекомендательная система на реальных данных.

При решении практических задач, если размер матрицы рейтингов не может быть легко умещен на одном компьютере, то использование реализации в Spark будет более практичным решением. Если существует

заинтересованность в применении рекомендательных систем более высокого уровня, то стоит подумать о гибридной системе, которая включает в себя информацию о пользователях/предметах вместе с историей покупки. Библиотека Python с названием LightFM выглядит очень интересно для такого рода приложений.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Collaborative Filtering for Implicit Feedback Datasets

[Электронный ресурс].URL: http://yifanhu.net/PUB/cf.pdf (дата обращения: 21 февраля 2017).

1. Архитектура рекомендательной системы, работающей на основе неявных пользовательских оценок [Электронный ресурс]. URL: http://ceur-ws.org/Vol-

803/paper8.pdf (дата обращения: 28 февраля 2017).

1. Implicit Recommender Systems: Biased Matrix Factorization [Электронный ресурс].URL:http://activisiongamescience.github.io/2016/01/11/Implicit-

Recommender-Systems-Biased-Matrix-Factorization/(дата обращения: 13 марта 2017).

1. Alternating Least Squares [Электронный ресурс].URL: https://datasciencemadesimpler.wordpress.com/tag/alternating-least-squares/ (дата обращения: 21 марта 2017).
2. Nonnegative Matrix Factorization and Recommendor Systems [Электронный ресурс]. URL: https://www.r-bloggers.com/nonnegative-matrix-factorization-andrecommendor-systems/ (дата обращения: 3 апреля 2017).
3. Рекомендательные системы: user-based и item-based [Электронный ресурс]. URL: https://habrahabr.ru/company/surfingbird/blog/139518/ (дата обращения: 18 апреля 2017).
4. Как работают рекомендательные системы. Лекция в Яндексе [Электронный ресурс]. URL: https://habrahabr.ru/company/yandex/blog/241455/

(дата обращения: 5 мая 2017).

1. How to build recommend system [Электронный ресурс]. URL:

http://blogs.gartner.com/martin-kihn/how-to-build-a-recommender-system-in-python/ (дата обращения: 17 мая 2017).

1. Introduction to approaches and algorithms [Электронный ресурс]. URL: https://www.ibm.com/developerworks/library/os-recommender1/ (дата обращения:

22 мая 2017).

10. Recommender Systems [Электронный ресурс]. URL: http://ijcai-

13.org/files/tutorial\_slides/td3.pdf (дата обращения: 23 мая 2017).