Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени первого Президента России Б. Н. Ельцина»

УТВЕРЖДАЮ

Директор по образовательной деятельности

_____ С.Т. Князев
«7» сентября 2023 г.

Рекомендательные системы

Учебно-методические материалы по направлению подготовки
09.03.03 Прикладная информатика
Образовательная программа «Прикладной искусственный интеллект»

Екатеринбург

РАЗРАБОТЧИКИ УЧЕБНО-МЕТОДИЧЕСКИХ МАТЕРИАЛОВ

Доцент Базовой кафедры «Аналитика больших данных и методы видеоанализа»	_____	М.А. Медведева
Ассистент Базовой кафедры «Аналитика больших данных и методы видеоанализа»	_____	Д. Балунгу

СОДЕРЖАНИЕ

Введение

За последние несколько десятилетий, с появлением Youtube, Amazon, Netflix и многих других подобных веб-сервисов, системы рекомендаций стали занимать все больше места в нашей жизни. Начиная с электронной коммерции (предлагая покупателям статьи, которые могут их заинтересовать) и заканчивая рекламой в Интернете (предлагая пользователям правильное содержание, соответствующее их предпочтениям), рекомендательные системы сегодня неизбежны в наших ежедневных онлайн-путешествиях.

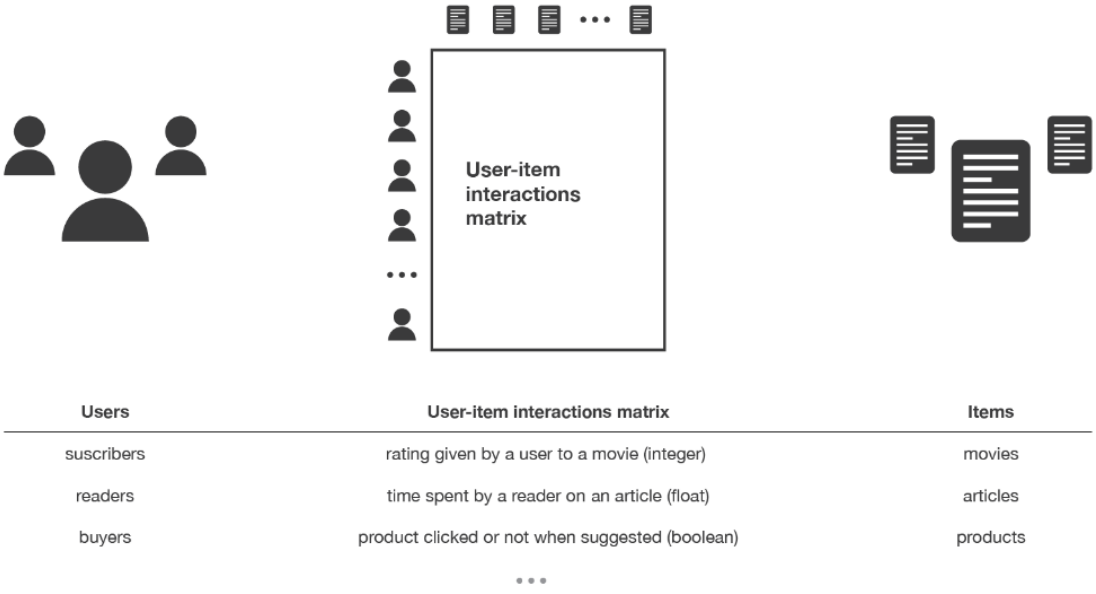
В общих чертах, рекомендательные системы — это алгоритмы, предназначенные для предложения пользователям соответствующих предметов (предметов, которые можно смотреть в фильмах, текста для чтения, продуктов для покупки или чего-либо еще, в зависимости от отрасли).

Рекомендательные системы действительно важны в некоторых отраслях, поскольку они могут приносить огромную прибыль, когда они эффективны, или также могут значительно отличаться от конкурентов. В качестве доказательства важности рекомендательных систем мы можем упомянуть, что несколько лет назад Netflix организовал испытание («приз Netflix»), где целью было создать рекомендательную систему, которая работает лучше, чем ее собственный алгоритм с призом. 1 миллион долларов, чтобы выиграть.

Рассмотрим различные парадигмы рекомендательных систем. Для каждого из них мы представим, как они работают, опишем их теоретические основы и обсудим их сильные и слабые стороны.

1 Методы совместной фильтрации

Совместные методы для рекомендательных систем — это методы, которые основаны исключительно на прошлых взаимодействиях, зарегистрированных между пользователями и предметами, с целью выработки новых рекомендаций. Эти взаимодействия хранятся в так называемой «матрице взаимодействий пользователь-элемент».



Users	User-item interactions matrix	Items
suscribers	rating given by a user to a movie (integer)	movies
readers	time spent by a reader on an article (float)	articles
buyers	product clicked or not when suggested (boolean)	products
...		

Рисунок 1 – Иллюстрация матрицы взаимодействий пользователь-элемент

Основная идея, которая управляет совместными методами, заключается в том, что этих прошлых взаимодействий пользователь-элемент достаточно для обнаружения аналогичных пользователей и / или аналогичных элементов и прогнозирования на основе этих предполагаемых приближений.

Класс алгоритмов совместной фильтрации разделен на две подкатегории, которые обычно называются основанными на памяти и модельными подходами. Подходы, основанные на памяти, напрямую работают со значениями записанных взаимодействий, предполагая отсутствие модели, и, по существу, основаны на поиске ближайших соседей (например, находят ближайших пользователей от интересующего пользователя и предлагают наиболее популярные элементы среди этих соседей). Подходы, основанные на моделях, предполагают базовую «порождающую» модель, которая объясняет взаимодействие пользователя с элементом и пытается обнаружить ее, чтобы делать новые предсказания.

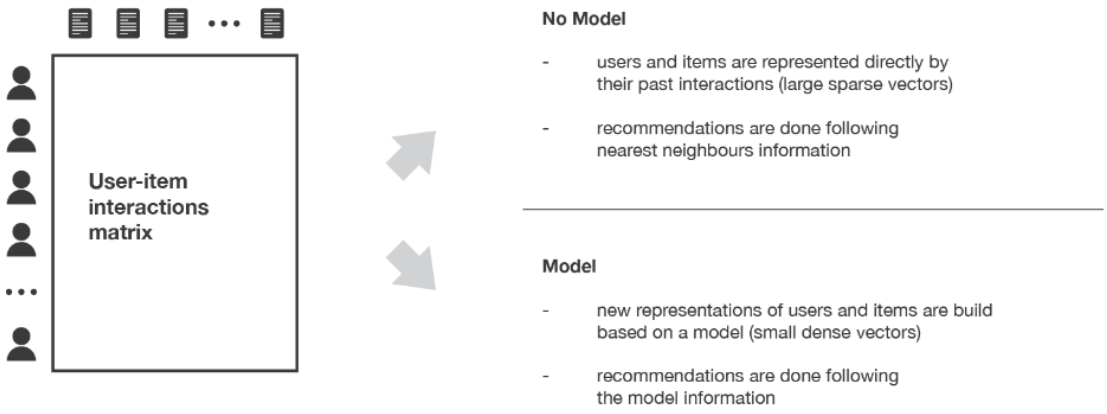


Рисунок 2 – Парадигмы методов совместной фильтрации

Основное преимущество совместных подходов заключается в том, что они не требуют информации о пользователях или элементах и, следовательно, могут использоваться во многих ситуациях. Более того, чем больше пользователей взаимодействуют с элементами, тем больше новых рекомендаций становятся точными: для фиксированного набора пользователей и элементов новые взаимодействия, записанные с течением времени, приносят новую информацию и делают систему более эффективной.

Тем не менее, поскольку для вынесения рекомендаций учитываются только прошлые взаимодействия, совместная фильтрация страдает от «проблемы холодного запуска»: невозможно рекомендовать что-либо новым пользователям или рекомендовать новый элемент любым пользователям, и многие пользователи или элементы имеют слишком мало взаимодействий. Быть эффективно обработанным. Этот недостаток можно устранить по-разному: рекомендовать случайные элементы новым пользователям или новые элементы случайным пользователям (случайная стратегия), рекомендовать популярные элементы новым пользователям или новые элементы наиболее активным пользователям (стратегия максимального ожидания), рекомендовать набор различных элементы для новых пользователей или новый элемент для набора различных пользователей (исследовательская стратегия) или, наконец, использование не совместного метода для ранней жизни пользователя или элемента.

В следующих разделах мы в основном представим три классических подхода к совместной фильтрации: два метода на основе памяти (пользователь-пользователь и элемент-элемент) и один подход на основе модели (матричная факторизация)

2 Контент-ориентированные методы

В отличие от методов совместной работы, которые полагаются только на взаимодействие элементов пользователя, подходы на основе контента используют дополнительную информацию о пользователях и / или элементах. Если мы рассмотрим пример системы рекомендации фильмов, такой дополнительной информацией может быть, например, возраст, пол, работа или любая другая личная информация для пользователей, а также категория, главные действующие лица, продолжительность или другие характеристики. Для фильмов (предметов).

Идея методов, основанных на контенте, состоит в том, чтобы попытаться построить модель, основанную на доступных «функциях», которые объясняют наблюдаемые взаимодействия пользователя с элементом. Все еще рассматривая пользователей и фильмы, мы попытаемся, например, смоделировать тот факт, что молодые женщины, как правило, оценивают лучше некоторые фильмы, что молодые мужчины, как правило, оценивают лучше некоторые другие фильмы и так далее. Если нам удастся получить такую модель, то сделать новые прогнозы для пользователя довольно просто: нам просто нужно посмотреть профиль (возраст, пол,…) этого пользователя и, основываясь на этой информации, определить соответствующие фильмы, чтобы предложить.

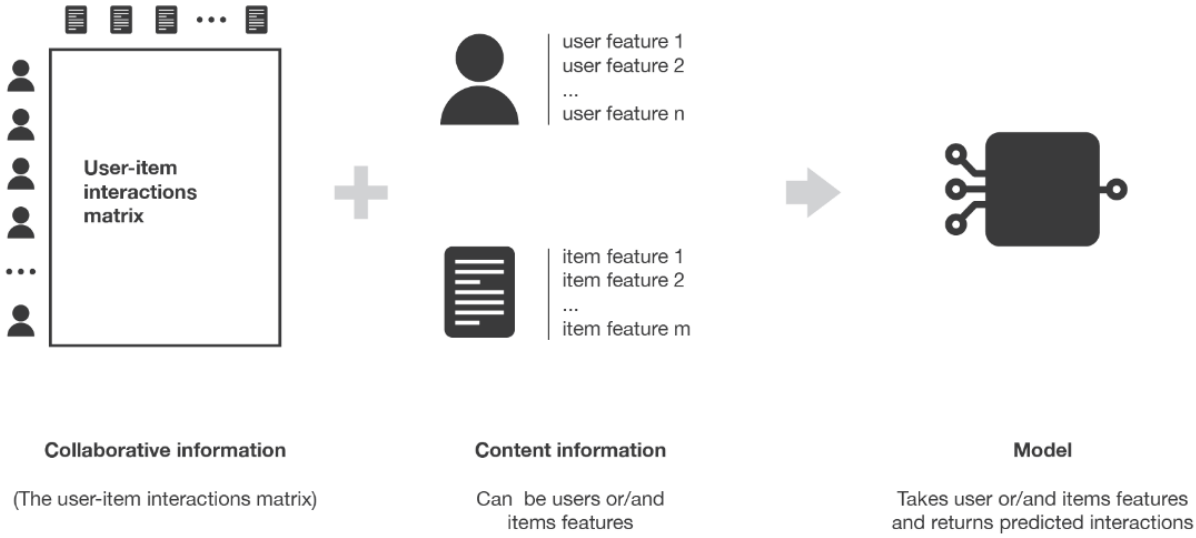


Рисунок 3 – Парадигма контент-ориентированных методов

Методы, основанные на контенте, гораздо меньше страдают от проблемы «холодного запуска», чем совместные подходы: новые пользователи или элементы могут быть описаны по их характеристикам (контенту), и поэтому могут быть сделаны соответствующие предложения для этих новых объектов. Этому недостатку логически

будут страдать только новые пользователи или элементы с ранее невиданными функциями, но как только система станет достаточно старой, у нее будет мало шансов вообще не случиться.

3 Сравнение совместных методов и контент-ориентированных методов

Рассмотрим, какое влияние имеет уровень моделирования на смещение и дисперсию.

В основанных на памяти совместных методах скрытая модель не предполагается. Алгоритмы напрямую работают с взаимодействиями пользователь-элемент: например, пользователи представлены своими взаимодействиями с элементами, и поиск ближайших соседей по этим представлениям используется для создания предложений. Поскольку скрытая модель не предполагается, эти методы теоретически имеют низкое смещение, но высокую дисперсию.

В основанных на модели методах сотрудничества предполагается некоторая модель скрытого взаимодействия. Модель обучена восстанавливать значения взаимодействий пользователь-элемент из собственного представления пользователей и элементов. Новые предложения могут быть сделаны на основе этой модели. Скрытые представления пользователей и элементов, извлеченные моделью, имеют математическое значение, которое может быть трудно интерпретировать для человека. Поскольку предполагается (довольно бесплатная) модель взаимодействия пользователя с элементом, эти методы теоретически имеют более высокий уклон, но более низкую дисперсию, чем методы, предполагающие отсутствие скрытой модели.

Наконец, в методах, основанных на контенте, также предполагается некоторая модель скрытого взаимодействия. Однако здесь модель снабжена контентом, который определяет представление пользователей и / или элементов: например, пользователи представлены заданными функциями, и мы пытаемся смоделировать для каждого элемента тип профиля пользователя, который нравится этому элементу или нет. Здесь, что касается основанных на модели методов взаимодействия, предполагается модель взаимодействия пользователя с элементом. Однако эта модель более ограничена (поскольку дано представление пользователей и / или элементов), и поэтому метод имеет тенденцию иметь самое высокое смещение, но самую низкую дисперсию.

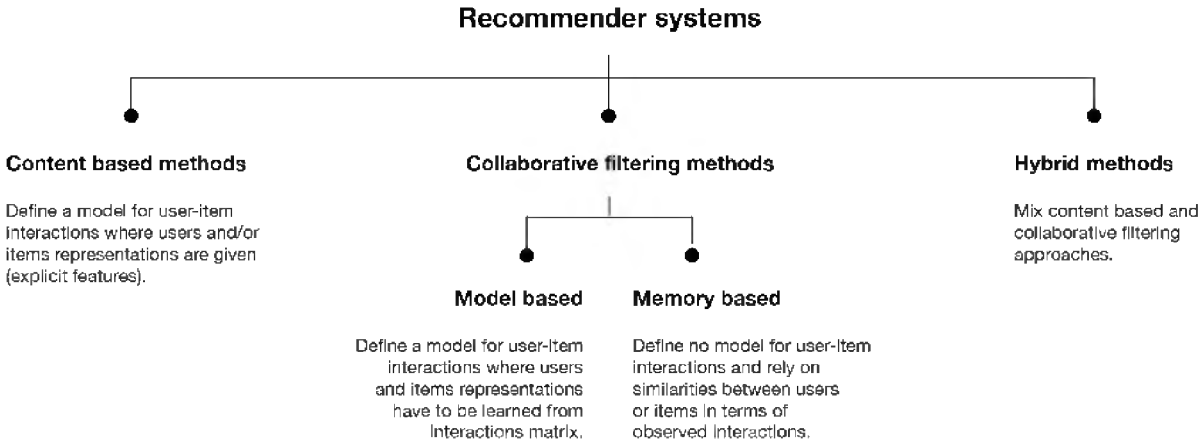


Рисунок 4 - Различные типы алгоритмов рекомендательных систем

4 Совместные методы на основе памяти

Основные характеристики пользователь-пользователь и элемент-элемент приближаются к тому, что они используют только информацию из матрицы взаимодействия элемент-пользователь и не предполагают модели для выработки новых рекомендаций.

4.1 Пользователь-пользователь

Чтобы дать новую рекомендацию пользователю, метод пользователь-пользователь приблизительно пытается идентифицировать пользователей с наиболее похожим «профилем взаимодействия» (ближайшими соседями), чтобы предложить элементы, которые являются наиболее популярными среди этих соседей (и которые « новое »для нашего пользователя). Этот метод называется «ориентированным на пользователя», поскольку он представляет пользователей на основе их взаимодействия с элементами и оценивает расстояния между пользователями.

Предположим, что мы хотим дать рекомендацию для данного пользователя. Во-первых, каждый пользователь может быть представлен своим вектором взаимодействия с различными элементами («его линия» в матрице взаимодействия). Затем мы можем вычислить какое-то «сходство» между интересующим нас пользователем и всеми остальными пользователями. Эта мера сходства такова, что два пользователя с похожим взаимодействием в одних и тех же элементах должны рассматриваться как близкие. После того, как были вычислены сходства для каждого пользователя, мы можем оставить k-ближайших соседей для нашего пользователя, а затем предложить наиболее популярные элементы среди них (только глядя на элементы, с которыми наш эталонный пользователь еще не взаимодействовал)

Обратите внимание, что при вычислении сходства между пользователями, число «общих взаимодействий» (сколько элементов уже было рассмотрено обоими пользователями?) Должно быть тщательно продумано! Действительно, большую часть времени мы хотим избежать того, чтобы кто-то, у кого есть только одно общее взаимодействие с нашим референтным пользователем, мог иметь 100% -ное совпадение и считаться «ближе», чем тот, кто имеет 100 общих взаимодействий и согласен «только» на 98% из них. Итак, мы считаем, что два пользователя похожи, если они взаимодействовали

с большим количеством общих элементов одинаковым образом (аналогичный рейтинг, подобное зависание времени…).

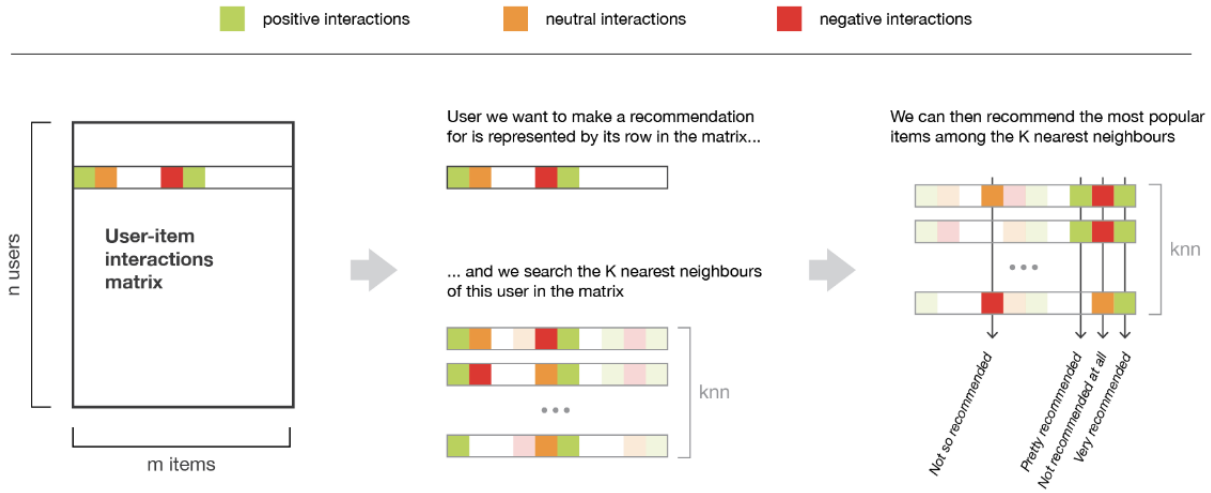


Рисунок 5 - Иллюстрация пользовательского метода

4.2 Item-Item

Чтобы дать новую рекомендацию пользователю, идея метода item-item состоит в том, чтобы находить элементы, похожие на те, с которыми пользователь уже «положительно» взаимодействовал. Два элемента считаются похожими, если большинство пользователей, которые взаимодействовали с ними обоими, делали это одинаково. Этот метод называется «центрированным на элементах», поскольку он представляет элементы на основе взаимодействий, которые пользователи имели с ними, и оценивает расстояния между этими элементами.

Предположим, что мы хотим дать рекомендацию для данного пользователя. Сначала мы рассмотрим элемент, который понравился этому пользователю больше всего, и представим его (как и все остальные элементы) по вектору взаимодействия с каждым пользователем («его столбец» в матрице взаимодействия). Затем мы можем вычислить сходство между «лучшим предметом» и всеми остальными предметами. После того, как сходства были вычислены, мы можем оставить k-ближайших соседей для выбранного «лучшего элемента», которые являются новыми для нашего интересующего пользователя, и рекомендовать эти элементы.

Обратите внимание, что для получения более релевантных рекомендаций мы можем выполнить эту работу не только для избранного элемента пользователя, а вместо этого рассмотрим n предпочтительных элементов. В этом случае мы можем порекомендовать товары, близкие к нескольким из этих предпочтительных.

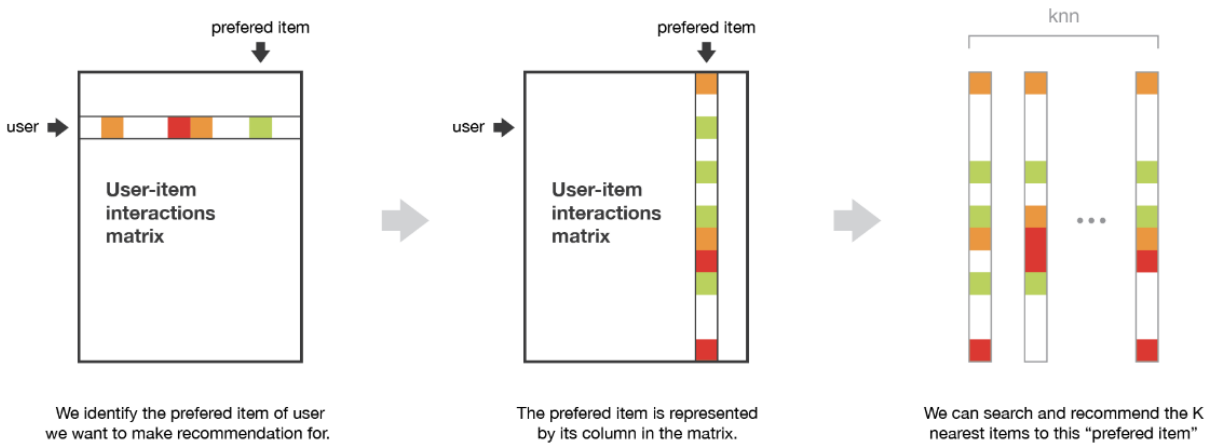


Рисунок 6 - Иллюстрация метода item-item

4.3 Сравнение user-user и item-item

Метод пользователь-пользователь основан на поиске похожих пользователей с точки зрения взаимодействия с элементами. Поскольку, как правило, каждый пользователь взаимодействовал только с несколькими элементами, это делает метод довольно чувствительным к любым зарегистрированным взаимодействиям (высокая дисперсия). С другой стороны, поскольку окончательная рекомендация основана только на взаимодействиях, зарегистрированных для пользователей, подобных интересующим нас пользователям, мы получаем более персонализированные результаты (низкий уклон).

И наоборот, метод элемент-элемент основан на поиске похожих элементов в терминах взаимодействий элемент-элемент. Поскольку, как правило, многие пользователи взаимодействуют с элементом, поиск окрестностей гораздо менее чувствителен к отдельным взаимодействиям (более низкая дисперсия). Как аналог, взаимодействия, исходящие от всех типов пользователей (даже пользователей, очень отличающихся от нашего эталонного пользователя), затем рассматриваются в рекомендации, что делает метод менее персонализированным (более предвзятым). Таким образом, этот подход менее персонализирован, чем подход пользователя и пользователя, но более надежен.

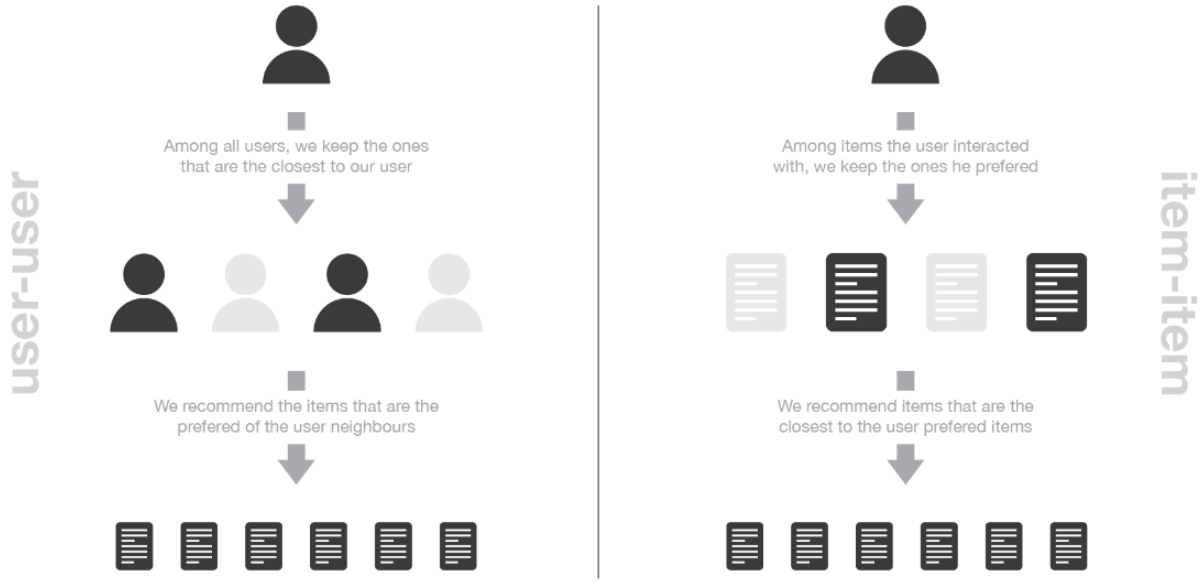


Рисунок 7 - Иллюстрация различий между методами item-item и user-user

4.4 Минусы методов на основе памяти

Один из самых больших недостатков совместной фильтрации на основе памяти заключается в том, что они не легко масштабируются: генерация новой рекомендации может быть чрезвычайно трудоемкой для больших систем. Действительно, для систем с миллионами пользователей и миллионами элементов этап поиска ближайших соседей может стать трудоемким, если не будет тщательно спроектирован (алгоритм KNN имеет сложность O (ndk) с n числом пользователей, d количеством элементов и k количество рассматриваемых соседей). Чтобы сделать вычисления более удобными для огромных систем, мы можем воспользоваться преимуществами разреженности матрицы взаимодействия при разработке нашего алгоритма или использовать приближенные методы ближайших соседей (ANN).

В большинстве рекомендательных алгоритмов необходимо быть предельно осторожным, чтобы избежать эффекта «богатый - получить - более богатый» для популярных предметов, и чтобы пользователи не попали в так называемую «область ограничения информации». Другими словами, мы не хотим, чтобы наша система рекомендовала все больше и больше только популярных товаров, а также мы не хотим, чтобы наши пользователи получали рекомендации только по товарам, очень близким к тому, который им уже нравился, без возможности узнать об этом. новые предметы, которые им тоже могут понравиться (поскольку эти предметы «недостаточно близки», чтобы их предлагать). Если, как мы упоминали, эти проблемы могут возникать в большинстве

рекомендательных алгоритмов, это особенно верно для основанных на памяти совместных алгоритмов. Действительно, из-за отсутствия модели «регуляризации» этот вид явления может усиливаться и наблюдаться чаще.

5 Совместные методы, основанные на модели

Совместные подходы, основанные на модели, основаны только на информации о взаимодействиях пользователя и элемента и предполагают, что скрытая модель должна объяснить эти взаимодействия. Например, алгоритмы матричной факторизации состоят в том, чтобы разложить огромную и разреженную матрицу взаимодействия пользователя с элементом в произведение двух меньших и плотных матриц: матрицы фактора пользователя (содержащей представления пользователей), которая умножает матрицу фактора-элемента (содержащую представления элементов).

5.1 Матричная факторизация

Основное предположение, лежащее в основе факторизации матрицы, состоит в том, что существует довольно низкоразмерное скрытое пространство признаков, в котором мы можем представлять, как пользователей, так и элементы, и что взаимодействие между пользователем и элементом может быть получено путем вычисления точечного произведения соответствующих плотных векторов. в этом пространстве.

Например, предположим, что у нас есть матрица рейтинга фильмов пользователя. Чтобы смоделировать взаимодействие между пользователями и фильмами, мы можем предположить, что:

- есть некоторые особенности, которые описывают (и рассказывают отдельно) довольно хорошие фильмы.
- эти функции также можно использовать для описания пользовательских предпочтений (высокие значения для функций, которые нравятся пользователю, низкие значения в противном случае)

Однако мы не хотим явно передавать эти функции нашей модели (как это можно сделать для подходов, основанных на контенте, которые мы опишем позже). Вместо этого мы предпочитаем позволить системе самостоятельно обнаруживать эти полезные функции и создавать собственные представления как пользователей, так и элементов. Поскольку они изучены и не даны, извлеченные признаки, взятые по отдельности, имеют математическое значение, но не имеют интуитивной интерпретации (и, следовательно, их трудно, если не невозможно, понять как человека). Тем не менее, нередко бывает, что структуры, возникающие из алгоритма такого типа, чрезвычайно близки к интуитивному разложению, о котором может думать человек. Действительно, следствием такой факторизации является то, что близкие пользователи с точки зрения предпочтений, а также близкие предметы с

точки зрения характеристик в конечном итоге имеют близкие представления в скрытом пространстве.

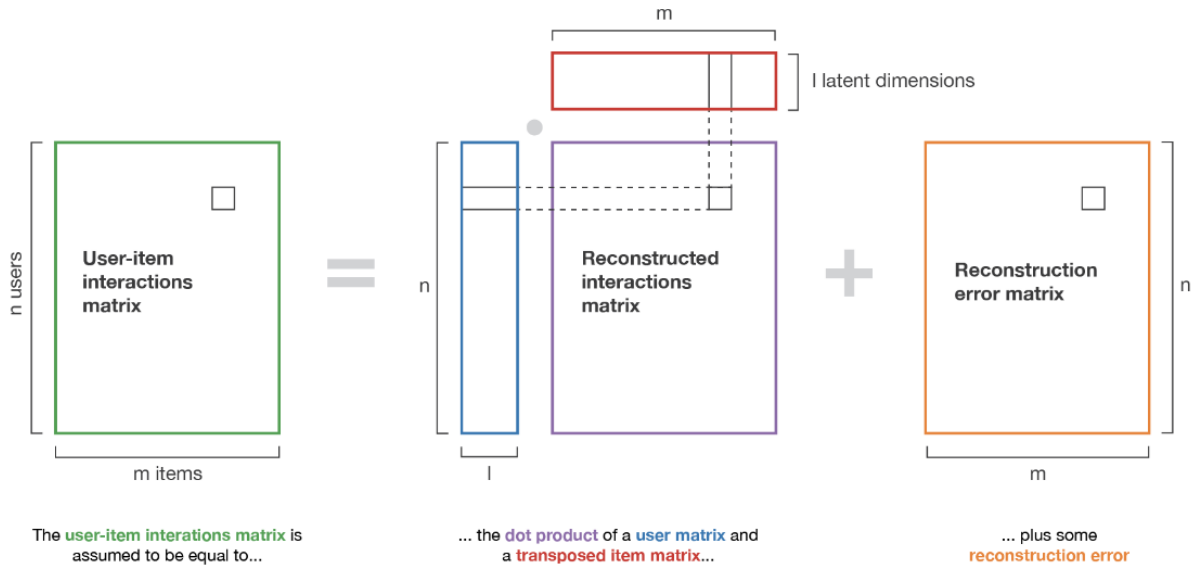


Рисунок 8 - Иллюстрация метода матричной факторизации

5.2 Математика матричной факторизации

В этом подразделе мы дадим простой математический обзор факторизации матрицы. В частности, мы описываем классический итерационный подход, основанный на градиентном спуске, который позволяет получать факторизации для очень больших матриц без одновременной загрузки всех данных в память компьютера.

Давайте рассмотрим матрицу взаимодействия M (nxm) рейтингов, в которой каждый пользователь оценивал только некоторые элементы (для большинства взаимодействий установлено значение Нет, чтобы выразить отсутствие оценки). Мы хотим разложить эту матрицу так, чтобы

$$M \approx X.Y^T,$$

где X - «пользовательская матрица» (nxl), строки которой представляют n пользователей, а где Y - «матрица элементов» (mxl), строки которой представляют m элементов:

$$user_i \equiv X_i \qquad \forall i \in \{1, ..., n\}$$
$$item_j \equiv Y_j \qquad \forall j \in \{1, ..., m\}$$

Здесь l - размер скрытого пространства, в котором будут представлены пользователи и элемент. Итак, мы ищем матрицы X и Y, произведение которых наилучшим образом приближает существующие взаимодействия. Обозначив E ансамбль пар (i, j) таким, что M_ij установлен (а не None), мы хотим найти X и Y, которые минимизируют «ошибку восстановления рейтинга»

$$(X, Y) = \underset{X,Y}{argmin} \sum_{(i,j)\in E} [(X_i)(Y_j)^T - M_{ij}]^2$$

Добавив коэффициент регуляризации и разделив на 2, получим

$$(X, Y) = \underset{X,Y}{argmin} \frac{1}{2} \sum_{(i,j)\in E} [(X_i)(Y_j)^T - M_{ij}]^2 + \frac{\lambda}{2}(\sum_{i,k}(X_{ik})^2 + \sum_{j,k}(Y_{jk})^2)$$

Матрицы X и Y затем могут быть получены в результате процесса оптимизации градиентного спуска, для которого мы можем заметить две вещи. Во-первых, градиент не нужно вычислять по всем парам в E на каждом шаге, и мы можем рассмотреть только подмножество этих пар, чтобы оптимизировать нашу целевую функцию «по пакетам». Во-вторых, значения в X и Y не должны обновляться одновременно, и градиентный спуск может быть выполнен поочередно для X и Y на каждом шаге (при этом мы считаем одну матрицу фиксированной и оптимизируем для другой, прежде чем делать обратное на следующем итерация).

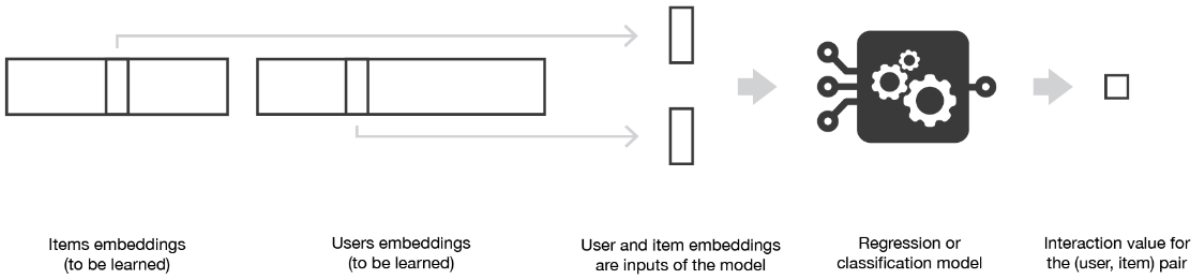
После того, как матрица была разложена, у нас остается меньше информации для манипуляции, чтобы дать новую рекомендацию: мы можем просто умножить вектор пользователя на любой вектор элемента, чтобы оценить соответствующий рейтинг. Обратите внимание, что мы могли бы также использовать методы user-user и item-item с этими новыми представлениями пользователей и элементов: (приблизительный) поиск ближайших соседей будет выполняться не по огромным разреженным векторам, а по маленьким плотным векторам, что делает некоторые методы аппроксимации более удобными для отслеживания.

5.3 Расширения матричной факторизации

Наконец, мы можем заметить, что концепция этой базовой факторизации может быть распространена на более сложные модели, например, с более общей нейронной сетью, такой как «декомпозиция» (мы больше не можем строго говорить о «факторизации»). Первая прямая адаптация, о которой мы можем думать, касается логических взаимодействий. Если мы хотим восстановить логические взаимодействия, простой точечный продукт не очень хорошо адаптирован. Однако, если мы добавим логистическую функцию поверх этого точечного произведения, мы получим модель, которая принимает значение в [0, 1], и, таким образом, лучше подходит для задачи. В таком случае модель для оптимизации

$$(X, Y) = \underset{X,Y}{argmin} \frac{1}{2} \sum_{(i,j)\in E} [f((X_i)(Y_j)^T) - M_{ij}]^2 + \frac{\lambda}{2}(\sum_{i,k}(X_{ik})^2 + \sum_{j,k}(Y_{jk})^2)$$

с f (.) нашей логистической функцией. Более глубокие модели нейронных сетей часто используются для достижения практически современных характеристик в сложных рекомендательных системах.



Матричная факторизация может быть обобщена с использованием модели поверх пользователей и вложений элементов.

6 Контент-ориентированные методы

В предыдущих двух разделах мы в основном обсуждали подходы пользователь-пользователь, элемент-элемент и матричная факторизация. Эти методы учитывают только матрицу взаимодействия пользователь-элемент и, таким образом, относятся к парадигме совместной фильтрации. Давайте теперь опишем основанную на контенте парадигму.

6.1 Концепция контент-методов

В методах, основанных на содержании, проблема рекомендации сводится либо к проблеме классификации (предсказать, нравится ли пользователю элемент, а не к элементу), либо к проблеме регрессии (предсказать оценку, присвоенную пользователю элементу). В обоих случаях мы собираемся установить модель, которая будет основана на возможностях пользователя и / или элемента в нашем распоряжении («контент» нашего «контентно-ориентированного» метода).

Если наша классификация (или регрессия) основана на особенностях пользователей, мы говорим, что подход ориентирован на элементы: моделирование, оптимизация и вычисления могут выполняться «по элементам». В этом случае мы строим и изучаем одну модель по элементам на основе функций пользователей, пытаясь ответить на вопрос «какова вероятность того, что каждому пользователю понравится этот элемент?» (Или «какова скорость, которую каждый пользователь дает этому элементу? Для регрессии). Модель, связанная с каждым элементом, естественно обучается на данных, связанных с этим элементом, и, как правило, приводит к довольно надежным моделям, так как многие пользователи взаимодействовали с элементом. Однако взаимодействия, рассматриваемые для изучения модели, исходят от каждого пользователя, и даже если эти пользователи имеют сходные характеристики (характеристики), их предпочтения могут отличаться. Это означает, что даже если этот метод является более надежным, его можно рассматривать как менее персонализированный (более предвзятый), чем метод, ориентированный на пользователя в дальнейшем.

Если мы работаем с функциями элементов, тогда метод ориентирован на пользователя: моделирование, оптимизация и вычисления могут выполняться «пользователем». Затем мы обучаем одну модель для каждого пользователя на основе функций элементов, которые пытаются ответить на вопрос «какова вероятность того, что этот пользователь полюбит каждый элемент?» (Или «какова скорость, которую этот пользователь дает каждому элементу?», Для регрессии). Затем мы можем прикрепить модель к каждому пользователю, обученному его данным: полученная модель, таким образом, более персонализирована, чем ее предметно-ориентированный аналог, поскольку она учитывает только взаимодействия от рассматриваемого пользователя. Однако в

большинстве случаев пользователь взаимодействовал с относительно небольшим количеством элементов, и поэтому полученная нами модель гораздо менее надежна, чем модель, ориентированная на элементы.

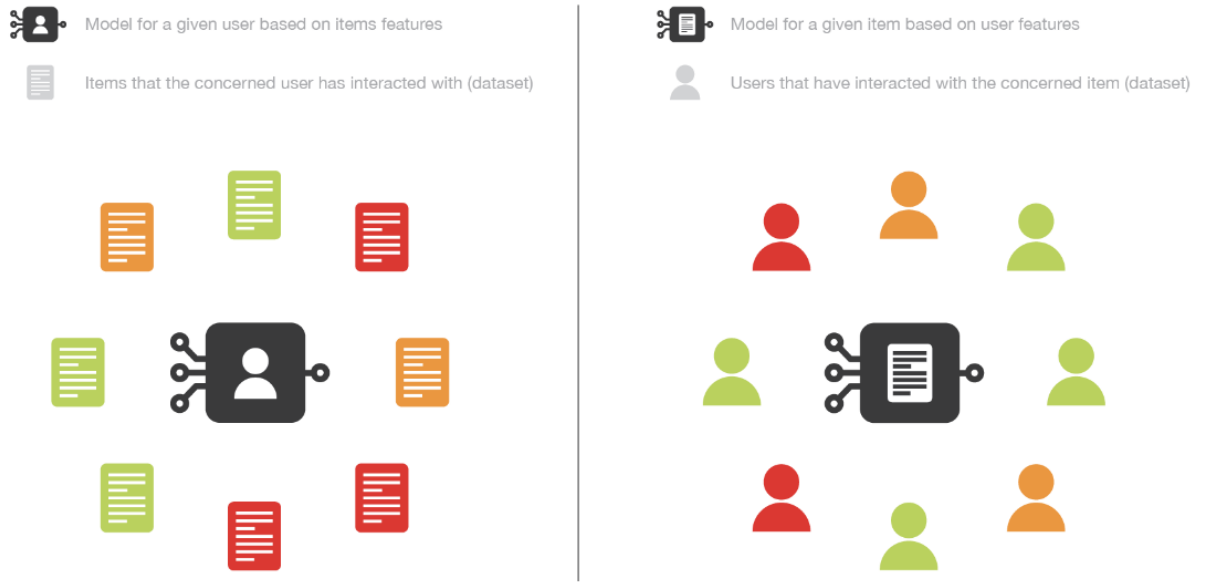


Рисунок 8 - Иллюстрация разницы между предметно-ориентированными и ориентированными на пользователя методами

С практической точки зрения, мы должны подчеркнуть, что в большинстве случаев гораздо сложнее задать какую-то информацию новому пользователю (пользователи не хотят отвечать на слишком много вопросов), чем задать много информации о новом пользователе. элемент (люди, добавляющие их, заинтересованы в заполнении этой информации, чтобы их элементы были рекомендованы нужным пользователям). Мы также можем заметить, что в зависимости от сложности отношения к выражению построенная нами модель может быть более или менее сложной, начиная от базовых моделей (логистическая / линейная регрессия для классификации / регрессии) до глубоких нейронных сетей. Наконец, отметим, что методы, основанные на контенте, также не могут быть ориентированы ни на пользователя, ни на элемент: для наших моделей можно использовать как информацию о пользователе, так и элемент, например, путем объединения двух векторов признаков и осуществления их через архитектуру нейронной сети.

6.2 Предметно-ориентированный байесовский классификатор

Давайте сначала рассмотрим случай предметно-ориентированной классификации: для каждого элемента мы хотим обучить байесовский классификатор, который принимает пользовательские функции в качестве входных данных и выводит «нравится» или «не нравится». Итак, чтобы выполнить задачу классификации, мы хотим вычислить

$$\frac{\mathbb{P}_{item}(like|user\_features)}{\mathbb{P}_{item}(dislike|user\_features)}$$

соотношение между вероятностью того, что пользователю с заданными характеристиками понравится рассматриваемый элемент, и вероятностью его неприязни. Это соотношение условных вероятностей, которое определяет наше правило классификации (с простым порогом), может быть выражено по формуле Байеса.

$$\mathbb{P}_{item}(like|user\_features) = \frac{\mathbb{P}_{item}(user\_features|like) \times \mathbb{P}_{item}(like)}{\mathbb{P}_{item}(user\_features)}$$

$$\mathbb{P}_{item}(dislike|user\_features) = \frac{\mathbb{P}_{item}(user\_features|dislike) \times \mathbb{P}_{item}(dislike)}{\mathbb{P}_{item}(user\_features)}$$

$$\frac{\mathbb{P}_{item}(like|user\_features)}{\mathbb{P}_{item}(dislike|user\_features)} = \frac{\mathbb{P}_{item}(user\_features|like) \times \mathbb{P}_{item}(like)}{\mathbb{P}_{item}(user\_features|dislike) \times \mathbb{P}_{item}(dislike)}$$

где

$$\mathbb{P}_{item}(like) \qquad \text{and} \qquad \mathbb{P}_{item}(dislike)(= 1 - \mathbb{P}_{item}(like))$$

Предполагается, что вероятности следуют гауссовским распределениям с параметрами, определяемыми также по данным. Можно сделать различные гипотезы о ковариационных матрицах этих двух распределений правдоподобия (без предположения, равенство матриц, равенство матриц и независимость признаков), приводящих к различным хорошо известным моделям (квадратичный дискриминантный анализ, линейный дискриминантный анализ, наивный байесовский классификатор). Мы можем еще раз подчеркнуть, что здесь параметры правдоподобия должны оцениваться только на основе данных (взаимодействий), связанных с рассматриваемым элементом.

Рисунок 9 - Иллюстрация предметно-ориентированного контента на основе байесовского классификатора

6.3 Ориентированная на пользователя линейная регрессия

Давайте теперь рассмотрим случай регрессии, ориентированной на пользователя: для каждого пользователя мы хотим обучить простую линейную регрессию, которая принимает характеристики элемента в качестве входных данных и выводит оценку для этого элемента. Мы по-прежнему обозначаем M как матрицу взаимодействия пользователя с элементом, мы складываем в матрицу X вектор-строки, представляющие пользовательские коэффициенты, которые необходимо выучить, и мы встраиваем в матрицу Y векторы-строки, представляющие элементы, которые даны. Затем для данного пользователя i мы изучаем коэффициенты в X_i, решая следующую задачу оптимизации

$$X_i = \underset{X_i}{argmin} \frac{1}{2} \sum_{(i,j) \in E} [(X_i)(Y_j)^T - M_{ij}]^2 + \frac{\lambda}{2} (\sum_k (X_{ik})^2)$$

где следует иметь в виду, что i фиксировано, и, таким образом, первое суммирование выполняется только по парам (пользователь, элемент), которые относятся к пользователю i. Мы можем заметить, что если мы решаем эту проблему для всех пользователей одновременно, то проблема оптимизации точно такая же, как и та, которую мы решаем в «чередовании матричных разрядов», когда мы сохраняем элементы фиксированными. Это наблюдение подчеркивает связь, которую мы упомянули в первом разделе: подходы коллективной фильтрации на основе модели (такие как факторизация матрицы) и методы, основанные на контенте, предполагают скрытую модель для взаимодействия пользователя с элементом, но подходы на основе модели, основанные на сотрудничестве, должны изучать скрытые представления для обоих пользователей. и элементы, в то время как методы, основанные на контенте, строят модель на основе определенных пользователем характеристик для пользователей и / или элементов.

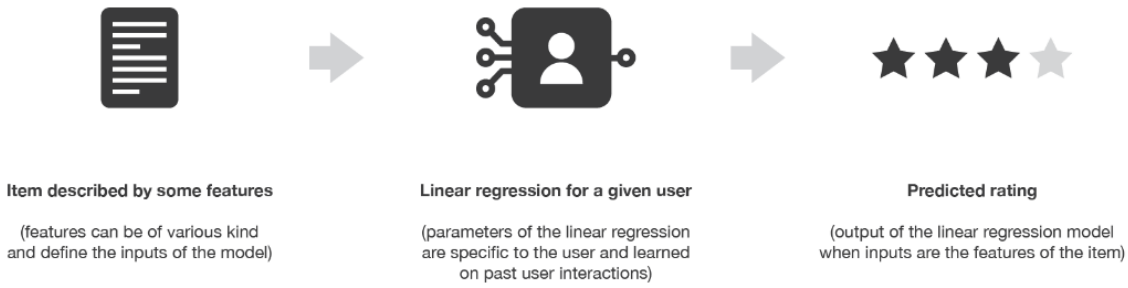


Рисунок 10 - Иллюстрация регрессии, ориентированной на пользователя

7 Оценка рекомендательной системы

Как и для любого алгоритма машинного обучения, мы должны уметь оценивать производительность наших рекомендательных систем, чтобы решить, какой алгоритм лучше всего подходит для нашей ситуации. Методы оценки для рекомендательных систем в основном можно разделить на два набора: оценка, основанная на четко определенных показателях, и оценка, в основном основанная на человеческом суждении и оценке удовлетворенности.

7.1 Оценка на основе метрик

Метрики для оценки рекомендательных систем

Средняя абсолютная ошибка (MAE)

$$MAE = (\frac{1}{n}) \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

Это наиболее простой показатель оценки, известный как средняя абсолютная ошибка. Вышеупомянутое представляет собой причудливое уравнение для его оценки. Это буквально разница между тем, как пользователь может оценить фильм, и тем, что предсказывает наша система.

Среднеквадратичная ошибка (RMSE)

$$RMSE = \sqrt{(\frac{1}{n}) \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

Это еще один распространенный и, пожалуй, самый популярный показатель оценки. Одна из причин заключается в том, что он меньше наказывает вас, когда вы приближаетесь к фактическому прогнозу, и гораздо больше, когда он далек от фактического прогноза, по сравнению с MAE.

На самом деле, мы действительно не оцениваем какие-либо современные рекомендательные системы, основанные только на точности. Системе рекомендательных систем наплевать, как бы пользователь оценил определенный фильм. Для рекомендательных систем важно то, что они помещают перед пользователями в список лучших рекомендателей, и как эти пользователи реагируют на эти фильмы, когда они видят

их рекомендованными. точность для оценки рекомендательных систем. Несмотря на то, что они присудили системе, Netflix не принял ее)

Итак, если система рекомендаций не фокусируется только на точности, что им делать?

Основная задача - **Top-N рекомендаций**, что означает, что работа рекомендательной системы состоит в том, чтобы создать конечный список лучших вещей, которые можно было бы представить определенному человеку.

Следующие показатели используются для оценки системы рекомендаций на основе рекомендаций Top-N.

Скорость попадания

$$HitRate = (\frac{hits}{users})$$

Это простая метрика. Во-первых, вы создаете для пользователя топ-N рекомендаций. Если одна из рекомендаций в верхних рекомендациях пользователя - это то, что он действительно оценил, вы считаете это хитом. Поскольку системе действительно удалось показать пользователю то, что он уже нашел достаточно интересным, чтобы посмотреть его самостоятельно, мы будем считать это успехом.

Поэтому для расчета мы складываем все совпадения в топ-N рекомендаций для каждого пользователя и делим их на каждого пользователя.

Средняя частота взаимных совпадений (ARHR)

$$ARHR = (\frac{1}{users}) \sum_{i=1}^{n} (1/rank_i)$$

Это вариация рейтинга попаданий, но она учитывает, в каком списке Top-N появляются ваши попадания. Таким образом, мы получаем больше очков за рекомендации предметов в верхнем слоте, чем в нижнем. Этот показатель больше ориентирован на пользователя. Если пользователю нужно прокрутить вниз, чтобы увидеть нижний элемент в вашем списке Top-N, имеет смысл наказывать рекомендацию, которая отображается слишком низко в списке, поскольку пользователю приходится работать, чтобы найти их.

Для рекомендательной системы есть еще несколько вещей. Теперь давайте посмотрим на них.

Покрытие

Проще говоря, это процент пар (пользователь, элемент), которые можно предсказать, или процент возможных рекомендаций, которые может предоставить рекомендательная система. Например, подумайте о наборе данных MovieLens для оценок фильмов. Он содержит рейтинги для нескольких тысяч фильмов, но существует множество фильмов, для которых у него нет оценок.

Поэтому, если мы используем эти данные для рекомендации фильмов на IMDB, который содержит несколько миллионов фильмов, охват будет довольно низким.

Стоит отметить, что покрытие может расходиться с точностью. Если вы установите более высокий порог качества для рекомендаций, которые вы делаете, вы можете повысить точность за счет покрытия.

Разнообразие

$$Diversity = (1 - S)\ where\ S = avg\ similarity\ between\ recommendation\ pairs$$

Подумайте об этой метрике, как о том, насколько широкий спектр элементов ваша рекомендательная система показывает пользователям.

Допустим, вы смотрите фильм о Джеймсе Бонде. Низкое разнообразие было бы рекомендательной системой, которая просто рекомендовала бы следующие части сериала о Джеймсе Бонде, но не рекомендует другие фильмы, которые не являются частью сериала о Джеймсе Бонде, но все же относятся к тому же жанру.

Очень большое разнообразие тоже не всегда хорошо. Совершенно случайные предметы имеют большое разнообразие, но это не очень хорошие рекомендации. Вам также необходимо проверить разнообразие наряду с некоторыми другими показателями, которые также измеряют качество рекомендаций.

Новинка

Новизна рекомендательных систем относится к тому, насколько популярны те товары, которые они рекомендуют. (т.е. средний рейтинг популярности рекомендуемых товаров)

И опять же, просто рекомендуя случайные вещи, вы получите очень высокие оценки новизны, поскольку подавляющее большинство товаров не являются самыми продаваемыми. Хотя новизна измерима, то, что с ней делать, во многом субъективно.

В рекомендательной системе есть концепция доверия пользователей. Люди хотят видеть в своих рекомендациях хотя бы несколько знакомых вещей.

Если мы будем рекомендовать только то, о чем люди никогда не слышали, они могут решить, что ваша система их не знает, и в результате они будут меньше интересоваться вашими рекомендациями. Кроме того, популярные вещи обычно популярны не зря. Они

нравятся значительной части населения, поэтому можно ожидать, что они станут хорошими рекомендациями для значительной части населения, которая их еще не читала и не смотрела.

7.2 Оценка на основе человека

При разработке системы рекомендаций мы можем быть заинтересованы не только в том, чтобы получить модель, которая дает рекомендации, в которых мы уверены, но мы также можем ожидать некоторые другие хорошие свойства, такие как разнообразие и объяснимость рекомендаций.

Как уже упоминалось в разделе для совместной работы, мы абсолютно не хотим, чтобы пользователь застрял в том, что мы называли ранее областью ограничения информации. Понятие «случайность» часто используется, чтобы выразить тенденцию, которую модель имеет или не создает такую область ограничения (разнообразие рекомендаций). Чтобы внести разнообразие в предлагаемый выбор, мы хотим порекомендовать товары, которые хорошо подходят как нашему пользователю, так и не слишком похожи друг на друга. Например, вместо того, чтобы рекомендовать пользователю «Начать войны» 1, 2 и 3, кажется, что лучше порекомендовать «Звездные войны 1», «Начать путь в темноту» и «Индиана Джонс и налетчики потерянного ковчега»: два - позже наша система может увидеть, что у нее меньше шансов заинтересовать пользователя, но рекомендовать 3 элемента, которые выглядят слишком похожими, не очень хороший вариант.

Объяснимость является еще одним ключевым моментом успеха алгоритмов рекомендации. Действительно, было доказано, что если пользователи не понимают, почему их рекомендовали в качестве конкретного элемента, они склонны терять доверие к системе рекомендаций. Таким образом, если мы создаем модель, которая легко объяснима, мы можем добавить, при вынесении рекомендаций, небольшое предложение, объясняющее, почему был рекомендован какой-либо элемент («людям, которым понравился этот элемент, также понравился этот элемент», «вам понравился этот элемент, вы может быть заинтересован этим »,…).

Наконец, в дополнение к тому факту, что разнообразие и объяснимость могут быть по сути сложными для оценки, мы можем заметить, что также довольно сложно оценить качество рекомендации, которая не относится к набору данных тестирования: как узнать, является ли новая рекомендация актуально, прежде чем рекомендовать его нашему пользователю? По всем этим причинам иногда бывает заманчиво протестировать модель в «реальных условиях». Поскольку целью системы рекомендаций является создание действия (просмотр

фильма, покупка продукта, чтение статьи и т. Д.), Мы действительно можем оценить ее способность генерировать ожидаемое действие. Например, система может быть запущена в производство в соответствии с подходом A / B-тестирования или может быть протестирована только на выборке пользователей. Однако такие процессы требуют определенного уровня доверия к модели.

Таким образом:

- Алгоритмы рекомендации можно разделить на две большие парадигмы:
- совместные подходы (такие как факторизация пользователя-пользователя, элемент-элемент и матрица), которые основаны только на матрице взаимодействия пользователя-элемента и подходы на основе контента (такие как модели регрессии или классификации), которые используют предварительная информация о пользователях и / или товарах.
- совместные методы, основанные на памяти, не предполагают какой-либо скрытой модели и имеют низкое смещение, но высокую дисперсию; основанные на модели коллективные подходы предполагают модель скрытых взаимодействий, которая должна изучать представления пользователей и элементов с нуля и, таким образом, иметь более высокий уклон, но меньшую дисперсию; методы, основанные на контенте, предполагают, что скрытая модель строится вокруг явно заданных функций пользователей и / или элементов и, таким образом, имеет наивысший уклон и наименьшую дисперсию
- Рекомендательные системы становятся все более и более важными во многих крупных отраслях промышленности, и при проектировании системы необходимо принимать во внимание некоторые масштабы (лучшее использование разреженности, итеративные методы для факторизации или оптимизации, приближенные методы поиска ближайших соседей…)
- системы рекомендаций трудно оценить: если можно использовать некоторые классические метрики, такие как MSE, точность, отзыв или точность, следует иметь в виду, что некоторые желаемые свойства, такие как разнородность и объяснимость, не могут быть оценены таким образом; оценка реальных условий (например, A / B-тестирование или тестирование образцов), наконец, является единственным реальным способом оценки новой рекомендательной системы, но требует определенной уверенности в модели

Гибридные подходы сочетают в себе методы совместной фильтрации и контент-ориентированного подхода, во многих случаях достигают самых современных результатов и поэтому используются в настоящее время во многих крупномасштабных

рекомендательных системах. Комбинация, сделанная в гибридных подходах, может в основном принимать две формы: мы можем либо обучить две модели независимо (одну модель совместной фильтрации и одну модель на основе контента) и объединить их предложения или напрямую построить единую модель (часто нейронную сеть), которая объединяет оба подхода, используя в качестве входных данных предварительную информацию (о пользователе и / или элементе), а также информацию о взаимодействии в сотрудничестве.

Лабораторная работа 1. Коллаборативная фильтрация на основе сходства по пользователям (user-based) и продуктам (item-based)

```
Словарь кинокритиков и выставленных ими оценок для небольшого набора
данных о фильмах
critics={'Lisa Rose': {'Lady in the Water': 2.5, 'Snakes on a Plane': 3.5,
 'Just My Luck': 3.0, 'Superman Returns': 3.5, 'You, Me and Dupree': 2.5,
 'The Night Listener': 3.0},
'Gene Seymour': {'Lady in the Water': 3.0, 'Snakes on a Plane': 3.5,
 'Just My Luck': 1.5, 'Superman Returns': 5.0, 'The Night Listener': 3.0,
 'You, Me and Dupree': 3.5},
'Michael Phillips': {'Lady in the Water': 2.5, 'Snakes on a Plane': 3.0,
 'Superman Returns': 3.5, 'The Night Listener': 4.0},
'Claudia Puig': {'Snakes on a Plane': 3.5, 'Just My Luck': 3.0,
 'The Night Listener': 4.5, 'Superman Returns': 4.0,
 'You, Me and Dupree': 2.5},
'Mick LaSalle': {'Lady in the Water': 3.0, 'Snakes on a Plane': 4.0,
 'Just My Luck': 2.0, 'Superman Returns': 3.0, 'The Night Listener': 3.0,
 'You, Me and Dupree': 2.0},
'Jack Matthews': {'Lady in the Water': 3.0, 'Snakes on a Plane': 4.0,
 'The Night Listener': 3.0, 'Superman Returns': 5.0, 'You, Me and Dupree':
3.5},
'Toby': {'Snakes on a Plane':4.5,'You, Me and Dupree':1.0,'Superman
Returns':4.0}}
```

```
import numpy as np
A=np.array(critics)
pd.DataFrame(critics)
```

	Lisa Rose	Gene Seymour	Michael Phillips	Claudia Puig	Mick LaSalle	Jack Matthews	Toby
Lady in the Water	2.5	3.0	2.5	NaN	3.0	3.0	NaN
Snakes on a Plane	3.5	3.5	3.0	3.5	4.0	4.0	4.5
Just My Luck	3.0	1.5	NaN	3.0	2.0	NaN	NaN
Superman Returns	3.5	5.0	3.5	4.0	3.0	5.0	4.0
You, Me and Dupree	2.5	3.5	NaN	2.5	2.0	3.5	1.0
The Night Listener	3.0						

```
critics['Toby']
```

```
critics['Lisa Rose']['Lady in the Water']
```

Вычисление расстояния Евклида

```
from math import sqrt
sqrt(pow(5-4,2)+pow(4-1,2))
```

Вычисление сходства

```
1/(1+sqrt(pow(5-4.5,2)+pow(5-5,2)))
from numpy import exp
exp(-0.3*sqrt(pow(5-4.5,2)+pow(5-5,2)))
from math import sqrt
```

Возвращает сходство person1 и person2 на основе расстояния

```
def sim_distance(prefs,person1,person2):
```

Получить список предметов, оцененных обоими

```
  si={}
  for item in prefs[person1]:
    if item in prefs[person2]:
      si[item]=1
```

Если нет ни одной общей оценки, вернуть 0

```
  if len(si)==0: return 0
```

Сложить квадраты разностей/

```
  sum_of_squares=sum([pow(prefs[person1][item]-prefs[person2][item],2)
  for item in prefs[person1] if item in prefs[person2]])
  return 1/(1+sum_of_squares)
from math import sqrt
```

Возвращает сходство person1 и person2 на основе расстояния

```
def sim_kernel(prefs,person1,person2,alpha=0.3):
```

Получить список предметов, оцененных обоими

```
  si={}
  for item in prefs[person1]:
    if item in prefs[person2]:
      si[item]=1
```

Если нет ни одной общей оценки, вернуть 0

```
    if len(si)==0: return 0
```
Сложить квадраты разностей
```
    sum_of_squares=sum([pow(prefs[person1][item]-prefs[person2][item],2)
    for item in prefs[person1] if item in prefs[person2]])
    return exp(-alpha*sum_of_squares)

sim_kernel(critics, 'Lisa Rose','Toby')
sim_distance(critics, 'Lisa Rose','Toby')
```
Возвращает коэффициент корреляции Пирсона между p1 и p2
```
def sim_pearson(prefs,p1,p2):
```
Получить список предметов, оцененных обоими
```
    si={}
    for item in prefs[p1]:
        if item in prefs[p2]: si[item]=1
```
Если нет ни одной общей оценки, вернуть 0
```
    if len(si)==0: return 0
```
Количество совместно оцененных фильм
```
    n=len(si)
```
Вычислить сумму всех предпочтений
```
    sum1=sum([prefs[p1][it] for it in si])
    sum2=sum([prefs[p2][it] for it in si])
```
Вычислить сумму квадратов
```
    sum1Sq=sum([pow(prefs[p1][it],2) for it in si])
    sum2Sq=sum([pow(prefs[p2][it],2) for it in si])
```
Вычислить сумму произведений
```
    pSum=sum([prefs[p1][it]*prefs[p2][it] for it in si])
```
Вычислить коэффициент Пирсона
```
    num=pSum-(sum1*sum2/n)
    den=sqrt((sum1Sq-pow(sum1,2)/n)*(sum2Sq-pow(sum2,2)/n))
    if den==0: return 0
```

```
    r=num/den
    return r
sim_pearson(critics,'Lisa Rose','Gene Seymour'), sim_distance(critics, 'Lisa Rose','Gene Seymour')
```

Ранжирование критиков

Возвращает список наилучших соответствий для человека из словаря prefs.

Количество результатов в списке и функция подобия – необязательные параметры.

```
def topMatches(prefs,person,n=5,similarity=sim_pearson):
    scores=[(similarity(prefs,person,other),other)
    for other in prefs if other!=person]
```

Отсортировать список по убыванию оценок

```
    scores.sort( )
    scores.reverse( )
    return scores[0:n]

topMatches(critics,'Toby',n=3, similarity=sim_distance)
topMatches(critics,'Toby',n=3, similarity=sim_kernel)
```

Рекомендация фильмов (User-based подход)

Получить рекомендации для заданного человека, пользуясь взвешенным средним оценок, данных всеми остальными пользователями

```
def getRecommendations(prefs,person,similarity=sim_pearson):
    totals={}
    simSums={}
    for other in prefs:
```

сравнивать меня с собой не нужно

```
        if other==person: continue
        sim=similarity(prefs,person,other)
```

игнорировать нулевые и отрицательные оценки

```
        if sim<=0: continue
        for item in prefs[other]:
```

оценивать только фильмы, которые я еще не смотрел

```
        if item not in prefs[person] or prefs[person][item]==0:
  Коэффициент подобия * Оценка
            totals.setdefault(item,0)
            totals[item]+=prefs[other][item]*sim
    Сумма коэффициентов подобия
            simSums.setdefault(item,0)
            simSums[item]+=sim
  Создать нормированный список
  rankings=[(total/simSums[item],item) for item,total in totals.items( )]
  Вернуть отсортированный список
  rankings.sort( )
  rankings.reverse( )
  return rankings

getRecommendations(critics,'Toby')
getRecommendations(critics,'Toby', sim_kernel)
```

Сходство предметов

Как заменить

```
{'Lisa Rose': {'Lady in the Water': 2.5, 'Snakes on a Plane': 3.5},
'Gene Seymour': {'Lady in the Water': 3.0, 'Snakes on a Plane': 3.5}}
```

на

```
{'Lady in the Water':{'Lisa Rose':2.5,'Gene Seymour':3.0},
'Snakes on a Plane':{'Lisa Rose':3.5,'Gene Seymour':3.5}}?

def transformPrefs(prefs):
  result={}
  for person in prefs:
```

```
    for item in prefs[person]:
      result.setdefault(item,{})
  Обменять местами человека и предмет
      result[item][person]=prefs[person][item]
  return result

movies=transformPrefs(critics)
movies
topMatches(movies,'Snakes on a Plane',5, sim_pearson)
getRecommendations(movies,'Lady in the Water', sim_distance)
```

Коллаборативная фильтрация по сходству объектов (Item-based collaborative filtering)

```
def calculateSimilarItems(prefs,n=10):
  Создать словарь, содержащий для каждого образца те образцы, которые
  больше всего похожи на него.
  result={}
  Обратить матрицу предпочтений, чтобы строки соответствовали образцам
  itemPrefs=transformPrefs(prefs)
  c=0
  for item in itemPrefs:
  Обновление состояния для больших наборов данных
    c+=1
    if c%100==0: print("%d / %d" % (c,len(itemPrefs)))
  Найти образцы, максимально похожие на данный
    scores=topMatches(itemPrefs,item,n=n,similarity=sim_distance)
    result[item]=scores
  return result

itemsim=calculateSimilarItems(critics)
itemsim
```

```
def getRecommendedItems(prefs,itemMatch,user):
  userRatings=prefs[user]
  scores={}
  totalSim={}

  Цикл по образцам, оцененным данным пользователем
  for (item,rating) in userRatings.items():

  Цикл по образцам, похожим на данный
    for (similarity,item2) in itemMatch[item]:
  Пропускаем, если пользователь уже оценивал данный образец
      if item2 in userRatings: continue
    Взвешенная суммы оценок, умноженных на коэффициент подобия
      scores.setdefault(item2,0)
      scores[item2]+=similarity*rating
    Сумма всех коэффициентов подобия
      totalSim.setdefault(item2,0)
      totalSim[item2]+=similarity
      if totalSim[item2]==0: totalSim[item2]=0.0000001 # чтобы избежать деления на ноль
  Делим каждую итоговую оценку на взвешенную сумму, чтобы вычислить среднее
  rankings=[(score/totalSim[item],item) for item,score in scores.items( ) ]

  Возвращает список rankings, отсортированный по убыванию
  rankings.sort( )
  rankings.reverse( )
  return rankings

getRecommendedItems(critics,itemsim,'Toby')
```

Рекомендация на данных MovieLens

Источник: http://grouplens.org/datasets/movielens/

```
def loadMovieLens(path='data/'):
```

Получить названия фильмов

```
  movies={}
  for line in open(path+'/u.item'):
    (id,title)=line.split('|')[0:2]
    movies[id]=title
```

Загрузить данные

```
  prefs={}
  for line in open(path+'/u.data'):
    (user,movieid,rating,ts)=line.split('\t')
    prefs.setdefault(user,{})
    prefs[user][movies[movieid]]=float(rating)
  return prefs
prefs=loadMovieLens( )
prefs['87']
len(prefs['87'])
getRecommendations(prefs,'87')[0:30]
itemsim=calculateSimilarItems(prefs,n=50)
itemsim["What's Eating Gilbert Grape (1993)"]
getRecommendedItems(prefs,itemsim,'87')[0:30]
```

Лабораторная работа 2. Матричная факторизация

1. Изучить теоретический материал по ссылке: http://activisiongamescience.github.io/2016/01/11/Implicit-Recommender-Systems-Biased-Matrix-Factorization/

2. Сравнить качество рекомендаций и время выполнения ALS (без базовых предикторов - without biases) и SGD (без базовых предикторов) для различных параметров регуляризации (например, 0,001, 0,01,...) и числа факторов (например, 1, 10, 25, 50,...). Для SGD необходимо усовершенствовать код.

3. Исправить код ALS так, чтобы была возможность производить расчеты с базовыми предикторами (biases).

4. Сравнить все 4 подхода по мере MSE, времени на обучение (training phase) и тестирование (inference phase). Результаты, помимо шагов выполнения в ipython, необходимо представить в сводной таблице.

Источник данных: MovieLens 100k. Может быть выбран альтернативный, но не меньший по числу оценок, либо пользователей и предметов (items).

Загружаем оценки и собираем из них матрицу оценок rij

[ ]

```
# загружаем данные.
names = ['user_id', 'item_id', 'rating', 'timestamp']
df = pd.read_csv('u.data', sep='\t', names=names)
n_users = df.user_id.unique().shape[0]
n_items = df.item_id.unique().shape[0]

# Создаем r_{ui} - нашу матрицу оценок
ratings = np.zeros((n_users, n_items))
for row in df.itertuples():
    ratings[row[1]-1, row[2]-1] = row[3]
```

Функция, которая делит множество оценок на test и train

[ ]

```
# Делим данные на train и test
# Для каждого пользователя 10 случайных оценок отправляем в test
def train_test_split(ratings):
    test = np.zeros(ratings.shape)
    train = ratings.copy()
    for user in np.arange(ratings.shape[0]):
        test_ratings = np.random.choice(ratings[user, :].nonzero()[0],
                                        size=10,
                                        replace=False)
        train[user, test_ratings] = 0.
        test[user, test_ratings] = ratings[user, test_ratings]

    # Проверяем, что  t est и train не пересекаются
```

```
    assert(np.all((train * test) == 0))
    return train, test
```

Метрика качества

```
[ ]
  from sklearn.metrics import mean_squared_error

  def get_mse(pred, actual):
    pred = pred[actual.nonzero()].flatten()
    actual = actual[actual.nonzero()].flatten()
    return mean_squared_error(pred, actual)
```

Класс, который реализует алгоритм ALS и ExplicitMF с градиентным спуском и смещениями

```
[ ]
  from numpy.linalg import solve


  class ExplicitMF():
    def __init__(self,
                 ratings,
                 n_factors=40,
                 learning='sgd',
                 item_fact_reg=0.0,
                 user_fact_reg=0.0,
                 item_bias_reg=0.0,
                 user_bias_reg=0.0,
                 verbose=False):
      """
      Train a matrix factorization model to predict empty
      entries in a matrix. The terminology assumes a
      ratings matrix which is ~ user x item

      Params
      ======
      ratings : (ndarray)
        User x Item matrix with corresponding ratings

      n_factors : (int)
        Number of latent factors to use in matrix
        factorization model
      learning : (str)
        Method of optimization. Options include
        'sgd' or 'als'.

      item_fact_reg : (float)
        Regularization term for item latent factors
```

```
    user_fact_reg : (float)
        Regularization term for user latent factors

    item_bias_reg : (float)
        Regularization term for item biases

    user_bias_reg : (float)
        Regularization term for user biases

    verbose : (bool)
        Whether or not to printout training progress
    """

    self.ratings = ratings
    self.n_users, self.n_items = ratings.shape
    self.n_factors = n_factors
    self.item_fact_reg = item_fact_reg
    self.user_fact_reg = user_fact_reg
    self.item_bias_reg = item_bias_reg
    self.user_bias_reg = user_bias_reg
    self.learning = learning
    if self.learning == 'sgd':
        self.sample_row, self.sample_col = self.ratings.nonzero()
        self.n_samples = len(self.sample_row)
    self._v = verbose

def als_step(self,
             latent_vectors,
             fixed_vecs,
             ratings,
             _lambda,
             type='user'):
    """
    One of the two ALS steps. Solve for the latent vectors
    specified by type.
    """
    if type == 'user':
        # Precompute
        YTY = fixed_vecs.T.dot(fixed_vecs)
        lambdaI = np.eye(YTY.shape[0]) * _lambda

        for u in range(latent_vectors.shape[0]):
            latent_vectors[u, :] = solve((YTY + lambdaI),
                                         ratings[u, :].dot(fixed_vecs))
    elif type == 'item':
        # Precompute
        XTX = fixed_vecs.T.dot(fixed_vecs)
        lambdaI = np.eye(XTX.shape[0]) * _lambda
```

```
        for i in range(latent_vectors.shape[0]):
            latent_vectors[i, :] = solve((XTX + lambdaI),
                                         ratings[:, i].T.dot(fixed_vecs))
    return latent_vectors

def train(self, n_iter=10, learning_rate=0.1):
    """ Train model for n_iter iterations from scratch."""
    # initialize latent vectors
    self.user_vecs = np.random.normal(scale=1./self.n_factors, size=(self.n_users, self.n_factors))
    self.item_vecs = np.random.normal(scale=1./self.n_factors, size=(self.n_items, self.n_factors))

    if self.learning == 'als':
        self.partial_train(n_iter,0)
    elif self.learning == 'sgd':
        self.learning_rate = learning_rate
        self.user_bias = np.zeros(self.n_users)
        self.item_bias = np.zeros(self.n_items)
        self.global_bias = np.mean(self.ratings[np.where(self.ratings != 0)])
        self.partial_train(n_iter,0)


def partial_train(self, n_iter, iter_done):
    """
    Train model for n_iter iterations. Can be
    called multiple times for further training.
    """
    ctr = 1
    while ctr <= n_iter:
        if (ctr+iter_done) % 10 == 0 and self._v:
            print (f'\tcurrent iteration: {ctr+iter_done}')
        if self.learning == 'als':
            self.user_vecs = self.als_step(self.user_vecs,
                                           self.item_vecs,
                                           self.ratings,
                                           self.user_fact_reg,
                                           type='user')
            self.item_vecs = self.als_step(self.item_vecs,
                                           self.user_vecs,
                                           self.ratings,
                                           self.item_fact_reg,
                                           type='item')
        elif self.learning == 'sgd':
            self.training_indices = np.arange(self.n_samples)
            np.random.shuffle(self.training_indices)
            self.sgd()
        ctr += 1

def sgd(self):
    for idx in self.training_indices:
        u = self.sample_row[idx]
```

```
            i = self.sample_col[idx]
            prediction = self.predict(u, i)
            e = (self.ratings[u,i] - prediction) # error

            # Update biases
            self.user_bias[u] += self.learning_rate * (e - self.user_bias_reg * self.user_bias[u])
            self.item_bias[i] += self.learning_rate * (e - self.item_bias_reg * self.item_bias[i])

            #Update latent factors
            self.user_vecs[u, :] += self.learning_rate *\
                            (e * self.item_vecs[i, :] - self.user_fact_reg * self.user_vecs[u,:])
            self.item_vecs[i, :] += self.learning_rate * \
                            (e * self.user_vecs[u, :] - self.item_fact_reg * self.item_vecs[i,:])
    def predict(self, u, i):
        """ Single user and item prediction."""
        if self.learning == 'als':
            return self.user_vecs[u, :].dot(self.item_vecs[i, :].T)
        elif self.learning == 'sgd':
            prediction = self.global_bias + self.user_bias[u] + self.item_bias[i]
            prediction += self.user_vecs[u, :].dot(self.item_vecs[i, :].T)
            return prediction

    def predict_all(self):
        """ Predict ratings for every user and item."""
        predictions = np.zeros((self.user_vecs.shape[0],
                                self.item_vecs.shape[0]))
        for u in range(self.user_vecs.shape[0]):
            for i in range(self.item_vecs.shape[0]):
                predictions[u, i] = self.predict(u, i)

        return predictions

    def calculate_learning_curve(self, iter_array, test, learning_rate=0.1):
        """
        Keep track of MSE as a function of training iterations.

        Params
        ======
        iter_array : (list)
            List of numbers of iterations to train for each step of
            the learning curve. e.g. [1, 5, 10, 20]
        test : (2D ndarray)
            Testing dataset (assumed to be user x item).

        The function creates two new class attributes:

        train_mse : (list)
            Training data MSE values for each value of iter_array
        test_mse : (list)
            Test data MSE values for each value of iter_array
```

```
        """
        iter_array.sort()
        self.train_mse =[]
        self.test_mse = []
        iter_diff = 0
        for (i, n_iter) in enumerate(iter_array):
          if self._v:
            print (f'Iteration: {n_iter}')
          if i == 0:
            self.train(n_iter - iter_diff, learning_rate)
          else:
            self.partial_train(n_iter - iter_diff, iter_diff)

          predictions = self.predict_all()

          self.train_mse += [get_mse(predictions, self.ratings)]
          self.test_mse += [get_mse(predictions, test)]
          if self._v:
            print (f'MSE train:test: {round(self.train_mse[-1],2)} : {round(self.test_mse[-1],2)}\n')
          iter_diff = n_iter
```

для построения графиков

[ ]

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
def plot_learning_curve(iter_array, model):
    plt.plot(iter_array, model.train_mse, \
             label='Training', linewidth=3)
    plt.plot(iter_array, model.test_mse, \
             label='Test', linewidth=3)
    plt.xticks(fontsize=16);
    plt.yticks(fontsize=16);
    plt.xlabel('Iterations', fontsize=25);
    plt.ylabel('MSE', fontsize=25);
    plt.legend(loc='best', fontsize=20);
```

Градиентный спуск с базовыми предикторами (biases)

[ ]

```
train, test = train_test_split(ratings)
MF_SGD = ExplicitMF(train, 40, learning='sgd', verbose=True)
iter_array = [1, 2, 5, 10, 25, 50, 100, 200]
```

[ ]

```
MF_SGD.calculate_learning_curve(iter_array, test, learning_rate=0.001)
```

Iteration: 1
MSE train:test: 1.14 : 1.19

Iteration: 2
MSE train:test: 1.07 : 1.14

Iteration: 5
MSE train:test: 0.98 : 1.06

Iteration: 10
current iteration: 10
MSE train:test: 0.92 : 1.01

Iteration: 25
current iteration: 20
MSE train:test: 0.87 : 0.96

Iteration: 50
current iteration: 30
current iteration: 40
current iteration: 50
MSE train:test: 0.84 : 0.94

Iteration: 100
current iteration: 60
current iteration: 70
current iteration: 80
current iteration: 90
current iteration: 100
MSE train:test: 0.75 : 0.92

Iteration: 200
current iteration: 110
current iteration: 120
current iteration: 130
current iteration: 140
current iteration: 150
current iteration: 160
current iteration: 170
current iteration: 180
current iteration: 190
current iteration: 200
MSE train:test: 0.4 : 0.92

[ ]
```
plot_learning_curve(iter_array, MF_SGD)
```

ALS без базовых предикторов

[ ]
```
MF_ALS = ExplicitMF(train, 40, learning='als', verbose=True)
iter_array = [1, 2, 5, 10, 25, 50, 100, 200]
MF_ALS.calculate_learning_curve(iter_array, test, learning_rate=0.001)
```
```
Iteration: 1
MSE train:test: 5.52 : 9.93

Iteration: 2
MSE train:test: 4.21 : 8.66

Iteration: 5
MSE train:test: 3.97 : 8.5

Iteration: 10
        current iteration: 10
MSE train:test: 3.93 : 8.48

Iteration: 25
        current iteration: 20
MSE train:test: 3.92 : 8.47

Iteration: 50
        current iteration: 30
        current iteration: 40
        current iteration: 50
MSE train:test: 3.92 : 8.48

Iteration: 100
        current iteration: 60
        current iteration: 70
        current iteration: 80
        current iteration: 90
        current iteration: 100
MSE train:test: 3.92 : 8.48

Iteration: 200
        current iteration: 110
        current iteration: 120
        current iteration: 130
        current iteration: 140
        current iteration: 150
        current iteration: 160
        current iteration: 170
        current iteration: 180
        current iteration: 190
        current iteration: 200
MSE train:test: 3.92 : 8.47
```

[ ]
```
plot_learning_curve(iter_array, MF_ALS)
```

[ ]

Лабораторная работа 3. Гибридная система рекомендаций с использованием Python

Приступим к созданию гибридной рекомендательной системы, импортировав необходимые библиотеки Python и **набор данных**:

```
import pandas as pd
data = pd.read_csv("fashion_products.csv")
print(data.head())
```

```
   User ID  Product ID Product Name   Brand         Category  Price    Rating  \
0       19           1        Dress  Adidas    Men's Fashion     40  1.043159
1       97           2        Shoes     H&M  Women's Fashion     82  4.026416
2       25           3        Dress  Adidas  Women's Fashion     44  3.337938
3       57           4        Shoes    Zara    Men's Fashion     23  1.049523
4       79           5      T-shirt  Adidas    Men's Fashion     79  4.302773

    Color Size
0   Black   XL
1   Black    L
2  Yellow   XL
3   White    S
4   Black    M
```

Таким образом, эти данные основаны на модных товарах для мужчин, женщин и детей. Наша цель — создать две системы рекомендаций с использованием совместной фильтрации и фильтрации на основе контента, а затем объединить методы рекомендаций для создания системы рекомендаций с использованием гибридного подхода.

Во-первых, давайте импортируем необходимые библиотеки Python, которые мы будем использовать для остальной части задачи:

```
from surprise import Dataset, Reader, SVD
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
```

В приведенном выше коде я импортировал библиотеку Surprise, которую вы, возможно, раньше не использовали. Библиотека сюрпризов импортирована для использования алгоритма SVD. SVD означает разложение по сингулярным значениям. Проще говоря, это метод матричной факторизации, обычно используемый в алгоритмах совместной фильтрации. Вы можете установить его в своих системах, используя команду, указанную ниже:

- Для терминала или командной строки: pip install scikit-surprise
- Для ноутбука Colab: !pip install scikit-surprise

Фильтрация на основе содержимого

Перейдем к созданию системы рекомендаций с использованием контентной фильтрации:

```
content_df = data[['Product ID', 'Product Name', 'Brand',
                   'Category', 'Color', 'Size']]
content_df['Content'] = content_df.apply(lambda row: '
'.join(row.dropna().astype(str)), axis=1)

# Use TF-IDF vectorizer to convert content into a matrix of TF-IDF features
tfidf_vectorizer = TfidfVectorizer()
content_matrix = tfidf_vectorizer.fit_transform(content_df['Content'])

content_similarity = linear_kernel(content_matrix, content_matrix)

reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(data[['User ID',
                                  'Product ID',
                                  'Rating']], reader)

def get_content_based_recommendations(product_id, top_n):
    index = content_df[content_df['Product ID'] == product_id].index[0]
    similarity_scores = content_similarity[index]
    similar_indices = similarity_scores.argsort()[::-1][1:top_n + 1]
    recommendations = content_df.loc[similar_indices, 'Product ID'].values
    return recommendations
```

В приведенном выше коде мы реализуем компонент фильтрации на основе контента гибридной рекомендательной системы. Мы начали с выбора соответствующих функций из набора данных, включая идентификатор продукта, название, бренд, категорию, цвет и размер. Затем мы объединили эти функции в один столбец «Контент» для каждого продукта.

Затем мы использовали векторизатор TF-IDF (Term Frequency-Inverse Document Frequency) для преобразования содержимого в матрицу признаков TF-IDF. Эта матрица представляет важность каждого слова в содержании по сравнению со всем корпусом.

Затем мы рассчитали сходство между продуктами на основе их содержания, используя косинусную меру сходства. Эта матрица сходства фиксирует сходство между каждой парой продуктов на основе их содержания.

Чтобы получить рекомендации на основе контента, мы сначала нашли индекс целевого продукта в матрице сходства. Затем мы отсортировали оценки сходства в порядке убывания и выбрали первые N похожих товаров. Наконец, мы вернули идентификаторы рекомендуемых продуктов.

Совместная фильтрация

Теперь давайте перейдем к созданию системы рекомендаций с использованием коллаборативной фильтрации:

```
algo = SVD()
trainset = data.build_full_trainset()
algo.fit(trainset)

def get_collaborative_filtering_recommendations(user_id, top_n):
    testset = trainset.build_anti_testset()
```

```
    testset = filter(lambda x: x[0] == user_id, testset)
    predictions = algo.test(testset)
    predictions.sort(key=lambda x: x.est, reverse=True)
    recommendations = [prediction.iid for prediction in predictions[:top_n]]
    return recommendations
```

В приведенном выше коде мы реализовали компонент совместной фильтрации гибридной рекомендательной системы с использованием алгоритма SVD (Singular Value Decomposition).

Во-первых, мы инициализировали алгоритм SVD и обучили его набору данных. Этот шаг включает в себя декомпозицию матрицы рейтинга пользовательских элементов, чтобы зафиксировать базовые шаблоны и скрытые факторы, определяющие пользовательские предпочтения.

Чтобы сгенерировать рекомендации по совместной фильтрации, мы создали тестовый набор, состоящий из пар «пользователь-элемент», которых не было в обучающем наборе. Мы отфильтровали этот тестовый набор, чтобы включить только элементы, принадлежащие целевому пользователю, указанному user_id.

Затем мы использовали обученную модель SVD для прогнозирования оценок элементов тестового набора. Эти прогнозы представляют собой предполагаемые рейтинги, которые пользователь назначит элементам.

Затем прогнозы сортируются по их оценочным рейтингам в порядке убывания. Мы выбрали первые N элементов с самыми высокими оценками в качестве рекомендаций по совместной фильтрации для пользователя.

Гибридный подход

Объединим методы контентной и коллаборативной фильтрации, чтобы построить рекомендательную систему с использованием гибридного метода:

```
def get_hybrid_recommendations(user_id, product_id, top_n):
    content_based_recommendations =
get_content_based_recommendations(product_id, top_n)
    collaborative_filtering_recommendations =
get_collaborative_filtering_recommendations(user_id, top_n)
    hybrid_recommendations = list(set(content_based_recommendations +
collaborative_filtering_recommendations))
    return hybrid_recommendations[:top_n]
```

В приведенном выше коде мы объединили подходы на основе контента и совместной фильтрации для создания гибридной рекомендательной системы.

Функция get_hybrid_recommendations принимает в качестве входных данных user_id, product_id и желаемое количество рекомендаций top_n.

Во-первых, он вызывает функцию get_content_based_recommendations, чтобы получить список рекомендаций на основе контента для указанного product_id. Эти рекомендации основаны на сходстве характеристик данного продукта с другими продуктами в наборе данных.

Затем он вызывает функцию get_collaborative_filtering_recommendations, чтобы получить список рекомендаций по совместной фильтрации для указанного user_id. Эти рекомендации генерируются путем использования исторических взаимодействий пользователя с элементом и оценки пользовательских предпочтений на основе аналогичного поведения пользователей.

Затем мы объединяем рекомендации по контентной и совместной фильтрации, взяв объединение двух списков. Он гарантирует, что гибридные рекомендации включают рекомендации по контенту и совместной фильтрации на основе пользовательских предпочтений.

Вот как можно использовать нашу гибридную систему рекомендаций, чтобы рекомендовать продукты на основе продукта, который просматривает пользователь:

```
user_id = 6
product_id = 11
top_n = 10
recommendations = get_hybrid_recommendations(user_id, product_id, top_n)

print(f"Hybrid Recommendations for User {user_id} based on Product
{product_id}:")
for i, recommendation in enumerate(recommendations):
    print(f"{i + 1}. Product ID: {recommendation}")
    print(f"{i + 1}. Product ID: {recommendation}")
```

```
Hybrid Recommendations for User 6 based on Product 11:
1. Product ID: 928
1. Product ID: 928
2. Product ID: 131
2. Product ID: 131
3. Product ID: 451
3. Product ID: 451
4. Product ID: 837
4. Product ID: 837
5. Product ID: 875
5. Product ID: 875
6. Product ID: 594
6. Product ID: 594
7. Product ID: 1463
7. Product ID: 1463
8. Product ID: 1688
8. Product ID: 1688
9. Product ID: 601
9. Product ID: 601
10. Product ID: 1566
10. Product ID: 1566
```

Итак, вот как создать гибридную систему рекомендаций с помощью Python. Гибридная система рекомендаций сочетает в себе несколько методов рекомендаций, чтобы предоставить пользователям более точные и разнообразные рекомендации. Он использует сильные стороны различных подходов, таких как совместная фильтрация и фильтрация на основе контента, чтобы преодолеть их ограничения и улучшить процесс рекомендаций.

Лабораторная работа 4. Контекстно-зависимые рекомендации корзины и персонализированные рекомендации

https://www.kaggle.com/code/shawamar/product-recommendation-system-for-e-commerce

Система рекомендаций по продуктам для предприятий электронной коммерции

Хорошо развитая система рекомендаций поможет компаниям улучшить качество обслуживания покупателей на веб-сайте и приведет к лучшему привлечению и удержанию клиентов.

Рекомендательная система основана на путешествии нового клиента с момента, когда он / она впервые попадает на веб-сайт компании, до момента, когда он / она совершает повторные покупки.

Рекомендательная система состоит из 3 частей в зависимости от бизнес-контекста:

1. **Рекомендательная система, часть I:** Система, ориентированная на новых клиентов
2. **Рекомендательная система, часть II:** Модельно-ориентированная система коллаборативной фильтрации, основанная на истории покупок клиентов и оценках, предоставленных другими пользователями, купившими аналогичные товары
3. **Рекомендательная система, часть III:** Когда бизнес впервые создает свой веб-сайт электронной коммерции без какого-либо рейтинга продукта

Когда новый клиент без какой-либо предыдущей истории покупок впервые посещает веб-сайт электронной коммерции, ему рекомендуются самые популярные товары, продаваемые на веб-сайте компании. После того, как он совершает покупку, рекомендательная система обновляет и рекомендует другие продукты на основе истории покупок и оценок, предоставленных другими пользователями на веб-сайте. Последняя часть выполняется с использованием методов коллаборативной фильтрации.

Рекомендательная система - Часть I¶

Рекомендательная система, основанная на популярности продукта, ориентированная на новых клиентов

1. Основанные на популярности являются отличной стратегией для нацеливания новых клиентов на самые популярные продукты, продаваемые на веб-сайте компании, и очень полезны для холодного запуска рекомендательной системы.
2. **Набор данных:** набор данных обзора продуктов Amazon

Импорт библиотек

```
import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt

# %matplotlib inline
plt.style.use("ggplot")

import sklearn
from sklearn.decomposition import TruncatedSVD
```

Загрузка набора данных

In [2]:

```
amazon_ratings = pd.read_csv('../input/amazon-ratings/ratings_Beauty.csv')
amazon_ratings = amazon_ratings.dropna()
amazon_ratings.head()
```

Out[2]:

	UserId	ProductId	Rating	Timestamp
0	A39HTATAQ9V7YF	0205616461	5.0	1369699200
1	A3JM6GV9MNOF9X	0558925278	3.0	1355443200
2	A1Z513UWSAAO0F	0558925278	5.0	1404691200
3	A1WMRR494NWEWV	0733001998	4.0	1382572800
4	A3IAAVS479H7M7	0737104473	1.0	1274227200

In [3]:

```
amazon_ratings.shape
```

Out[3]:

```
(2023070, 4)
```

In [4]:

```
popular_products = pd.DataFrame(amazon_ratings.groupby('ProductId')['Rating'].count()
)
most_popular = popular_products.sort_values('Rating', ascending=False)
most_popular.head(10)
```

Out[4]:

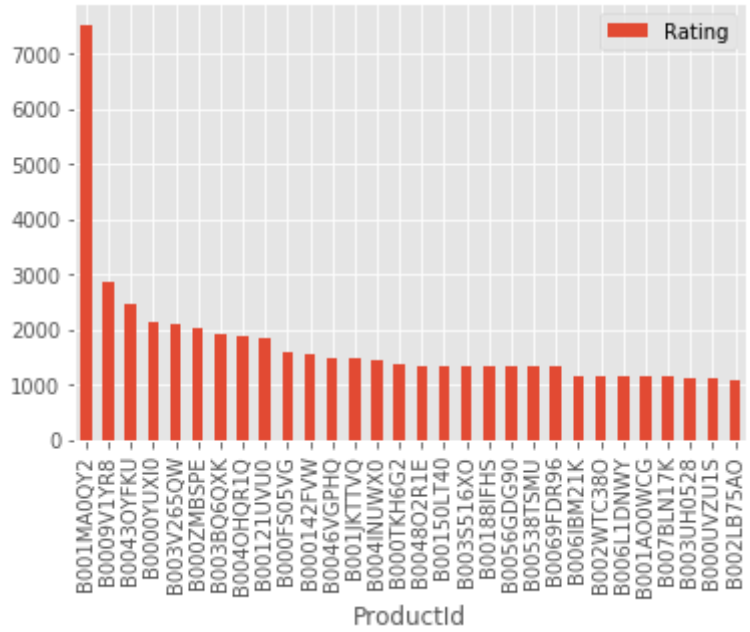
	Rating
ProductId	
B001MA0QY2	7533
B0009V1YR8	2869
B0043OYFKU	2477
B0000YUXI0	2143
B003V265QW	2088
B000ZMBSPE	2041
B003BQ6QXK	1918
B004OHQR1Q	1885
B00121UVU0	1838
B000FS05VG	1589

In [5]:

```
most_popular.head(30).plot(kind = "bar")
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd439e493c8>
```



Анализ:

1. На приведенном выше графике представлены самые популярные продукты (расположенные в порядке убывания), продаваемые бизнесом.
2. Например, продукт, ID # B001MA0QY2 имеет продажи более 7000, следующий по популярности продукт, ID # B0009V1YR8 имеет продажи 3000 и т. Д.

Рекомендательная система - Часть II

Система коллаборативной фильтрации на основе моделей

1. Рекомендуйте товары пользователям на основе истории покупок и сходства оценок, предоставленных другими пользователями, купившими товары, с оценками конкретного клиента.
2. Метод коллаборативной фильтрации на основе моделей здесь подходит, поскольку он помогает в создании продуктов прогнозирования для конкретного пользователя, выявляя закономерности на основе предпочтений из нескольких пользовательских данных.

Матрица полезности на основе проданных товаров и отзывов пользователей

Матрица полезности: Матрица utlity состоит из всех возможных деталей предпочтений (оценок) пользовательских элементов, представленных в виде матрицы. Матрица полезности скудна, так как ни один из пользователей не купит все элементы в списке, следовательно, большинство значений неизвестны.

In [6]:

```
# Subset of Amazon Ratings
```

```
amazon_ratings1 = amazon_ratings.head(10000)
```

In [7]:

```
ratings_utility_matrix = amazon_ratings1.pivot_table(values='Rating', index='UserId',
columns='ProductId', fill_value=0)
ratings_utility_matrix.head()
```

Out[7]:

ProductId	0205616461	0558925278	0733001998	0737104473	0762451459	1304139212	1304139220	130414089X	130414643X	1304146537	...	B00000 52YPE	B00000 52YPF	B00000 52YPG	B00000 52YPH	B00000 52YPM	B00000 52YPU	B00000 52YPV	B00000 52YPY	B00000 52YQ0	B00000 52YQ2
UserId																					
A0020592 1JHJK5X 9LNP42	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
A0245811 34CV80 ZBLIZTZ	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
A0305658 1JJIOL5 FSKJY7	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
A0309910 1ZRK4K 607JVHH	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

ProductId	0205616461	0558925278	0733001998	0737104473	0762451459	1304139212	1304139220	130414089X	130414643X	1304146537	...	B00005YPE	B00005YPF	B00005YPG	B00005YPH	B00005YPM	B00005YPU	B00005YPV	B00005YPY	B00005YQ0	B00005YQ2
UserId																					
A0505229A7NSH3FRXRR4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 строк × 886 столбцов

Как и ожидалось, матрица полезности, приведенная выше, скудна, я заполнил неизвестные значения с 0.

In [8]:

```
ratings_utility_matrix.shape
```

Out[8]:

```
(9697, 886)
```

Transposing the matrix

In [9]:

```
X = ratings_utility_matrix.T
X.head()
```

Out[9]:

UserId	A0020592 1JHJK5X9LNP42	A0245811 34CV80ZBLIZTZ	A0305658 1JJIOL5FSKJY7	A0309910 1ZRK4K607JVHH	A0505229 A7NSH3FRXRR4	A0549266 3T95KW63BR75K	A0595479 20Q3LZVFHLPI3	A0741023 2KYRFR25CIUGJ	A0827966 24UNM47DSAI6K	A0864963 DOAY7LXGS5I6	...	AZW1HXXYAC15B	AZWRTJPN7NXT	AZWTXHXZXFAYP	AZYQEFB9Y5N22	AZZHB6U54UDYW	AZZHJZP4GQPPZ	AZZNK89PXD006	AZZOFVMQC0BJG	AZZQXL8VDCFTV	AZZTJQ7CQZUD8
ProductId																					
0205616461	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0558925278	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0733001998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

UserId	A0020592 1JHJK5X9LNP42	A0245811 34CV80ZBLIZTZ	A0306581 JJIOL5FSKJY7	A0309910 1ZRK4K607JVHH	A0502229 A7NSH3FRXRR4	A0549266 3T95KW63BR75K	A0595479 20Q3LZVFHLPI3	A0741023 2KYRFR25CIUGJ	A0827966 24UNM47DSAI6K	A0864963 DOAY7LXGS5I6	...	AZW1HXXYAC15B	AZWRTJPN7NXT	AZWTXHXZXFAYP	AZYQEFB9Y5N22	AZZHB6U54UDYW	AZZHJZP4GQPPZ	AZZNK89PXD006	AZZOFVMQC0BJG	AZZQXL8VDCFTV	AZZTJQ7CQZUD8
ProductId																					
0737104473	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0762451459	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 строк × 9697 столбцов

In [10]:

```
X.shape
```

Out[10]:

```
(886, 9697)
```

Unique products in subset of data

In [11]:
```
X1 = X
```
Decomposing the Matrix

In [12]:
```
SVD = TruncatedSVD(n_components=10)
decomposed_matrix = SVD.fit_transform(X)
decomposed_matrix.shape
```

Out[12]:
```
(886, 10)
```
Correlation Matrix

In [13]:
```
correlation_matrix = np.corrcoef(decomposed_matrix)
correlation_matrix.shape
```

Out[13]:
```
(886, 886)
```
correlation_matrix

Isolating Product ID # 6117036094 from the Correlation Matrix

Assuming the customer buys Product ID # 6117036094 (randomly chosen)

In [14]:
```
X.index[99]
```

Out[14]:
```
'6117036094'
```
Index # of product ID purchased by customer

In [15]:
```
i = "6117036094"

product_names = list(X.index)
product_ID = product_names.index(i)
product_ID
```

Out[15]:
```
99
```
Корреляция для всех товаров с товаром, приобретенным этим клиентом, на основе товаров, оцененных другими покупателями, людьми, купившими тот же продукт

In [16]:
```
correlation_product_ID = correlation_matrix[product_ID]
correlation_product_ID.shape
```

Out[16]:
```
(886,)
```
Рекомендация 10 лучших продуктов с высокой степенью коррелирования в последовательности

In [17]:
```
Recommend = list(X.index[correlation_product_ID > 0.90])

Removes the item already bought by the customer
```

```
Recommend.remove(i)

Recommend[0:9]
```

Out[17]:

```
['0733001998',
 '1304139212',
 '1304139220',
 '130414089X',
 '130414643X',
 '130414674X',
 '1304174778',
 '1304174867',
 '1304174905']
```

Идентификатор продукта # Вот 10 лучших продуктов, которые будут отображаться системой рекомендаций вышеуказанному клиенту на основе истории покупок других клиентов на веб-сайте.

Рекомендательная система - Часть III

1. Для бизнеса, не имеющего истории покупок пользовательских товаров, для пользователей может быть разработана система рекомендаций на основе поисковых систем. Рекомендации по продукту могут быть основаны на анализе текстовой кластеризации, приведенном в описании продукта.

   Набор данных: набор данных Home Depot с набором данных о продукте.

In [18]:

```
# Importing libraries

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
```

Постатейная рекомендательная система на основе описания продукта

Применимо, когда бизнес впервые создает свой веб-сайт электронной коммерции

In [19]:

```
product_descriptions = pd.read_csv('../input/home-depot-product-search-relevance/prod
uct_descriptions.csv')
product_descriptions.shape
```

Out[19]:

```
(124428, 2)
```

Проверка отсутствующих значений

In [20]:

```
Отсутствующие значения

product_descriptions = product_descriptions.dropna()
product_descriptions.shape
product_descriptions.head()
```

Out[20]:

	product_uid	product_description
0	100001	Not only do angles make joints stronger, they ...
1	100002	BEHR Premium Textured DECKOVER is an innovativ...
2	100003	Classic architecture meets contemporary design...
3	100004	The Grape Solar 265-Watt Polycrystalline PV So...
4	100005	Update your bathroom with the Delta Vero Singl...

In [21]:

```
product_descriptions1 = product_descriptions.head(500)
# product_descriptions1.iloc[:,1]

product_descriptions1["product_description"].head(10)
```

Out[21]:

```
0    Not only do angles make joints stronger, they ...
1    BEHR Premium Textured DECKOVER is an innovativ...
2    Classic architecture meets contemporary design...
3    The Grape Solar 265-Watt Polycrystalline PV So...
4    Update your bathroom with the Delta Vero Singl...
5    Achieving delicious results is almost effortle...
6    The Quantum Adjustable 2-Light LED Black Emerg...
7    The Teks #10 x 1-1/2 in. Zinc-Plated Steel Was...
8    Get the House of Fara 3/4 in. x 3 in. x 8 ft. ...
9    Valley View Industries Metal Stakes (4-Pack) a...
Name: product_description, dtype: object
```

Извлечение признаков из описаний товаров

Преобразование текста в описании товара в числовые данные для анализа

In [22]:

```
vectorizer = TfidfVectorizer(stop_words='english')
X1 = vectorizer.fit_transform(product_descriptions1["product_description"])
X1
```

Out[22]:

```
<500x8932 sparse matrix of type '<class 'numpy.float64'>'
        with 34817 stored elements in Compressed Sparse Row format>
```

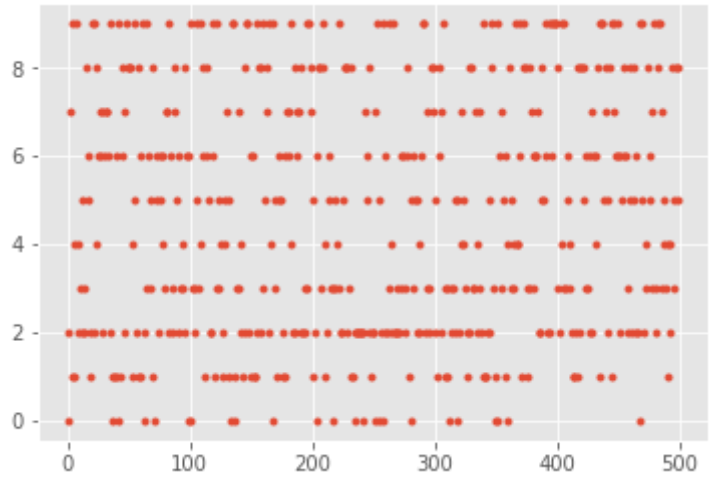
Визуализация кластеров продуктов в подмножестве данных

In [23]:

```
# Fitting K-Means to the dataset

X=X1

kmeans = KMeans(n_clusters = 10, init = 'k-means++')
y_kmeans = kmeans.fit_predict(X)
plt.plot(y_kmeans, ".")
plt.show()
```



F

In [24]:

```
def print_cluster(i):
    print("Cluster %d:" % i),
    for ind in order_centroids[i, :10]:
        print(' %s' % terms[ind]),
    print
```

1. Рекомендация продукта на основе текущего продукта, выбранного пользователем.
2. Рекомендовать сопутствующий товар можно исходя из того, что часто покупают вместе.

Лучшие слова в каждом кластере на основе описания продукта

In [25]:

Оптимальными кластерами являются

```
true_k = 10

model = KMeans(n_clusters=true_k, init='k-means++', max_iter=100, n_init=1)
model.fit(X1)

print("Top terms per cluster:")
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()
for i in range(true_k):
    print_cluster(i)
```

```
Top terms per cluster:
Cluster 0:
 concrete
```

```
 stake
 ft
 coating
 apply
 epoxy
 drying
 sq
 garage
 formula
Cluster 1:
 wood
 patio
 bamboo
 natural
 frame
 outdoor
 rug
 size
 steel
 dining
Cluster 2:
 used
 trim
 painted
 65
 proposition
 nbsp
 residents
 california
 project
 32
Cluster 3:
 door
 lbs
 easy
 dog
 nickel
 solid
 roof
 plastic
 house
 adjustable
Cluster 4:
 cutting
 saw
 tool
 blade
 design
 cut
 pliers
 grip
 metal
 non
Cluster 5:
 wall
 piece
 finish
 tile
```

```
 design
 use
 color
 easy
 installation
 water
Cluster 6:
 light
 watt
 bulb
 led
 fixture
 volt
 bulbs
 lighting
 use
 power
Cluster 7:
 helps
 water
 easy
 snow
 handle
 nozzle
 year
 features
 tool
 control
Cluster 8:
 air
 ft
 water
 unit
 room
 installation
 fan
 cooling
 use
 easy
Cluster 9:
 post
 fence
 gate
 ft
 screen
 vinyl
 posts
 aluminum
 brackets
 spline
```

Прогнозирование кластеров на основе ключевых поисковых слов

In [26]:

```
def show_recommendations(product):
    #print("Cluster ID:")
    Y = vectorizer.transform([product])
    prediction = model.predict(Y)
```

```
    #print(prediction)
    print_cluster(prediction[0])
```

- **Keyword :** режущий инструмент

In [27]:
```
show_recommendations("cutting tool")
```

```
Cluster 4:
 cutting
 saw
 tool
 blade
 design
 cut
 pliers
 grip
 metal
 non
```

Ключевое слово : аэрозольная краска

In [28]:
```
show_recommendations("spray paint")
```

```
Cluster 2:
 used
 trim
 painted
 65
 proposition
 nbsp
 residents
 california
 project
 32
```

Ключевое слово : стальное сверло

In [29]:
```
show_recommendations("steel drill")
```

```
Cluster 8:
 air
 ft
 water
 unit
 room
 installation
 fan
 cooling
 use
 easy
```

In case a word appears in multiple clusters, the algorithm chooses the cluster with the highest frequency of occurance of the word.

Ключевое слово : вода

In [30]:

```
show_recommendations("water")
```

```
Cluster 8:
 air
 ft
 water
 unit
 room
 installation
 fan
 cooling
 use
 easy
```

linkcode

После того, как кластер идентифицирован на основе поисковых слов пользователя, рекомендательная система может отображать элементы из соответствующих кластеров продуктов на основе описаний продуктов.

Примечание:

Это работает лучше всего, если компания впервые настраивает свой веб-сайт электронной коммерции и не имеет истории покупок / оценок пользовательских товаров, с которой можно было бы начать с самого начала. Эта рекомендательная система поможет пользователям получить хорошую рекомендацию для начала, и как только у покупателей будет история покупок, рекомендательный механизм может использовать метод коллаборативной фильтрации на основе модели.

Лабораторная работа 5 Разработка рекомендательной системы на Python (Memory-Based Collaborative Filtering и Model-Based Collaborative filtering)

Примечание: в контексте данной работы единицами рекомендаций будут являться фильмы, т.к. использование этого термина будет удобно с точки зрения используемого датасета.

Можно выделить два основных типа рекомендательных систем.

Content-based

- Пользователю рекомендуются фильмы, похожие на те, которые он уже посмотрел.
- Похожести оцениваются по признакам содержимого объектов.
- Сильная зависимость от предметной области, полезность рекомендаций ограничена.

Коллаборативная фильтрация (**Collaborative Filtering**)

- Для рекомендации используется история оценок как самого пользователя, так и других пользователей.
- Более универсальный подход, часто дает лучший результат.
- Есть свои проблемы (например, холодный старт).

В большинстве случаев алгоритмы коллаборативной фильтрации (CF) показывают лучший результат, чем content-based системы. В данной работе будут рассматриваться два типа CF: **Memory-Based Collaborative Filtering** и **Model-Based Collaborative filtering**.

Датасет

В данной работе используется MovieLens Dataset (Small). Посмотреть информацию или скачать датасет можно отсюда.

```
import numpy as np
import pandas as pd
# загружаем датасет
df = pd.read_csv('ml-latest-small/ratings.csv')
# смотрим на структуру
df.head()
```

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

```
# выводим количество пользователей и фильмов
n_users = df['userId'].unique().shape[0]
n_items = df['movieId'].unique().shape[0]

print('Users: {}, Items: {}'.format(n_users, n_items))

Users: 671, Items: 9066

# чтобы можно было удобно работать дальше, необходимо отмасштабировать
# значения в колонке movieId (новые значения будут в диапазоне от 1 до
```

```
# количества фильмов)
input_list = df['movieId'].unique()

def scale_movie_id(input_id):
    return np.where(input_list == input_id)[0][0] + 1

df['movieId'] = df['movieId'].apply(scale_movie_id)

from sklearn import cross_validation as cv

# делим данные на тренировочный и тестовый наборы
train_data, test_data = cv.train_test_split(df, test_size=0.20)
```

Memory-Based Collaborative Filtering

Memory-Based Collaborative Filtering подходы можно разделить на две части: **user-item filtering** and **item-item filtering**.

В user-item filtering мы:

1. Берём исходного пользователя.
2. Находим группу пользователей, которая максимально похожа на него (основываясь, например, оценках) и узнаём, какие фильмы понравились этой группе.
3. Нашему исходному пользователю рекомендуем фильмы, которые нравятся найденной группе пользователей.

На входе пользователь, на выходе – рекомендация фильмов для данного пользователя.

В item-item filtering мы:

1. Берём какой-либо фильм
2. Находим пользователей, которым нравится этот фильм
3. Смотрим на фильмы, которые нравятся найденным пользователям и выводим их в качестве рекомендации к исходному товару

На входе фильм, на выходе – рекомендация в виде похожих фильмов.

- Item-Item Collaborative Filtering: "Пользователям, которым нравится данный фильм, может так же понравиться это ..."
- User-Item Collaborative Filtering: "Похожим на вас пользователям нравится это ..."

В обоих случаях нам необходимо создать user-item матрицу, которая будет выглядеть следующим образом:

	Item1	Item2	Item3
User1	5	3	4
User2	4	0	0
User3	0	0	0

В ячейках матрицы будет записана информация об оценке фильма *m* пользователя *n*.

```
# создаём две user-item матрицы – для обучения и для теста
```

```
train_data_matrix = np.zeros((n_users, n_items))
for line in train_data.itertuples():
    train_data_matrix[line[1] - 1, line[2] - 1] = line[3]

test_data_matrix = np.zeros((n_users, n_items))
for line in test_data.itertuples():
    test_data_matrix[line[1] - 1, line[2] - 1] = line[3]
```

После того, как мы построим данную матрицу, на её основе необходимо будет рассчитать две новые матрицы с коэффициентами сходства (похожести, близости) для пользователей и для фильмов.

В качестве метрики близости в данной работе используется косинусное расстояние между векторами пользователей (фильмов).

Формула для пользователей:

$$s_u^{cos}(u_k, u_a) = \frac{u_k \cdot u_a}{\|u_k\| \, \|u_a\|} = \frac{\sum x_{k,m} x_{a,m}}{\sqrt{\sum x_{k,m}^2 \sum x_{a,m}^2}}$$

Формула для фильмов:

$$s_u^{cos}(i_m, i_b) = \frac{i_m \cdot i_b}{\|i_m\| \, \|i_b\|} = \frac{\sum x_{a,m} x_{a,b}}{\sqrt{\sum x_{a,m}^2 \sum x_{a,b}^2}}$$

```
from sklearn.metrics.pairwise import pairwise_distances

# считаем косинусное расстояние для пользователей и фильмов
user_similarity = pairwise_distances(train_data_matrix, metric='cosine')
item_similarity = pairwise_distances(train_data_matrix.T, metric='cosine')
```

Матрица "похожести" для пользователей имеет следующий вид (аналогичную структуру имеет и матрицы для фильмов):

	User1	User1	User3
User1	0	0.87	0.99
User2	123	0	123123
User3	123	123	0

Для предсказания в user-based CF необходимо применить следующую формулу:

$$\hat{x}_{k,m} = \overline{x}_k + \frac{\sum_{u_a} sim_u(u_k, u_a)(x_{a,m} - \overline{x}_{u_a})}{\sum_{u_a} |sim_u(u_k, u_a)|}$$

Для item-based CF:

$$\hat{x}_{k,m} = \frac{\sum_{i_b} sim_i(i_m, i_b)(x_{k,b})}{\sum_{i_b} |sim_i(i_m, i_b)|}$$

```
def predict(ratings, similarity, type='user'):
    if type == 'user':
        mean_user_rating = ratings.mean(axis=1)
        ratings_diff = (ratings - mean_user_rating[:, np.newaxis])
        pred = mean_user_rating[:, np.newaxis] + similarity.dot(ratings_diff)
/ np.array([np.abs(similarity).sum(axis=1)]).T
    elif type == 'item':
        pred = ratings.dot(similarity) /
np.array([np.abs(similarity).sum(axis=1)])
    return pred


item_prediction = predict(train_data_matrix, item_similarity, type='item')
user_prediction = predict(train_data_matrix, user_similarity, type='user')
```

Для оценки качества предсказания используем метрику RMSE (Root Mean Square Error, среднеквадратичная ошибка):

$$RMSE = \sqrt{\frac{1}{N}\sum(x_i - \hat{x}_i)^2}$$

```
from sklearn.metrics import mean_squared_error
from math import sqrt

def rmse(prediction, ground_truth):
    prediction = prediction[ground_truth.nonzero()].flatten()
    ground_truth = ground_truth[ground_truth.nonzero()].flatten()
    return sqrt(mean_squared_error(prediction, ground_truth))


print('User-based CF RMSE: ' + str(rmse(user_prediction, test_data_matrix)))
print('Item-based CF RMSE: ' + str(rmse(item_prediction, test_data_matrix)))
User-based CF RMSE: 3.3359002165272122
Item-based CF RMSE: 3.559460954114629
```

Model-based Collaborative Filtering

Model-based Collaborative Filtering основана на разложении матрицы. В данной работе используется метод разложения, который называется **singular value decomposition** (SVD, сингулярное разложение). Смысл этого разложения в том, что исходную матрицу X мы разбиваем на произведение ортогональных матриц U и V^T и диагональной матрицы S.

$$X = UV^T S$$

В нашем случае X – разреженная (состоящая преимущественно из нулей) user-item матрица. Разложив её на компоненты, мы можем их вновь перемножить и получить "восстановленную" матрицу X^. Матрица X^ и будет являться нашим предсказанием – метод SVD сделал сам за нас всё работу и заполнил пропуски в исходной матрице X.

$$UV^T S \approx \hat{X}$$

```
import scipy.sparse as sp
from scipy.sparse.linalg import svds
# делаем SVD
u, s, vt = svds(train_data_matrix, k=10)
s_diag_matrix = np.diag(s)

# предсказываем
X_pred = np.dot(np.dot(u, s_diag_matrix), vt)

# выводим метрику
print('User-based CF MSE: ' + str(rmse(X_pred, test_data_matrix)))
```

```
User-based CF MSE: 2.9952680217994416
```

Hybrid Recommender Systems

Пару слов стоит сказать о наиболее эффективной на практике гибридной рекомендательной системе. Его суть заключается в том, чтобы комбинировать в себе content-based модели и collaborative filtering. Используя дополнительную информацию о фильмах (жанр, теги, год выпуска и т.д.) и мощь алгоритмов коллаборативной фильтрации, можно добиться впечатляющего результата.

Что почитать по этой теме

Implementing your own recommender systems in Python
Как работают рекомендательные системы. Лекция в Яндексе

Контрольная работа

Примерный перечень тем

1.Рекомендательная система, основанная на коллаборативной фильтрации
2. Рекомендательная система, основанная на контенте
3. Рекомендательная система, основанная на знаниях

Примерные задания

1. Рекомендация элементов с использованием факторизации матрицы с функциями может (отметьте все, что применимо):
а) обеспечить персонализацию
б) захватить контекст (например, время суток)
в) ничего из вышеперечисленного
2. В какой из следующих ситуаций коллаборативная система фильтрации будет наиболее подходящим алгоритмом обучения (по сравнению с линейной или логистической регрессией)?
- Вы управляете книжным интернет-магазином, и у вас есть рейтинги книг от многих пользователей. Вы хотите научиться прогнозировать ожидаемый объем продаж (количество проданных книг) в зависимости от среднего рейтинга книги.
- Вы художник и рисуете портреты для своих клиентов. Каждый клиент получает свой портрет (себя) и дает вам обратную связь с рейтингом от 1 до 5 звезд, и каждый клиент покупает не более 1 портрета. Вы хотели бы предсказать, какую оценку даст вам ваш следующий клиент.
- Вы управляете книжным интернет-магазином и собираете рейтинги многих пользователей. Вы хотите использовать это, чтобы определить, какие книги «похожи» друг на друга (т. е. если одному пользователю нравится определенная книга, какие другие книги ему также могут понравиться?)
- У вас есть магазин одежды, в котором продаются джинсы многих стилей и марок. Вы собрали отзывы о различных стилях и брендах от частых покупателей, и вы хотите использовать эти обзоры, чтобы предложить этим покупателям скидки на джинсы, которые, по вашему мнению, они, скорее всего, приобретут.
- Вы написали программное обеспечение, которое загружает новостные статьи со многих новостных сайтов. В своей системе вы также отслеживаете, какие статьи вам лично нравятся, а какие нет, и система также хранит функции этих статей (например, количество слов, имя автора). Используя эту информацию, вы хотите построить систему, чтобы попытаться найти дополнительные новые статьи, которые лично вам понравятся.
- Вы запускаете онлайн-агрегатор новостей, и для каждого пользователя вы знаете некоторое подмножество статей, которые нравятся пользователю, и какое-то другое подмножество, которое не нравится пользователю. Вы захотите использовать это, чтобы найти другие статьи, которые нравятся пользователю.
- Вы управляете книжным интернет-магазином, и у вас есть рейтинги книг от многих пользователей. Для каждого пользователя вы хотите порекомендовать другие книги, которые ему понравятся, основываясь на его собственных оценках и оценках других пользователей.
3. Вы управляете киноимперией и хотите создать систему рекомендаций фильмов, основанную на коллаборативной фильтрации. Было три популярных обзорных веб-сайта (которые мы назовем A, B и C), на которые пользователи переходили, чтобы оценить фильмы, и вы только что приобрели все три компании, которые управляют этими веб-

сайтами. Вы хотели бы объединить наборы данных трех компаний, чтобы создать единую / унифицированную систему. На веб-сайте A пользователи оценивают фильм как имеющий от 1 до 5 звезд. На веб-сайте B пользователи ранжируются по шкале от 1 до 10, допускаются десятичные значения (например, 7,5). На сайте C рейтинги от 1 до 100. У вас также достаточно информации, чтобы идентифицировать пользователей/фильмы на одном веб-сайте с пользователями/фильмами на другом веб-сайте. Какое из следующих утверждений верно?

- Вы можете объединить три набора данных в один, но сначала вы должны нормализовать оценки каждого набора данных (скажем, изменить масштаб оценок каждого набора данных до диапазона 0-1).
- Вы можете объединить все три обучающих набора в один, если вы выполняете нормализацию среднего значения и масштабирование функций после объединения данных.
- Предполагая, что в одной базе данных есть хотя бы один фильм/пользователь, который также не отображается во второй базе данных, нет надежного способа объединить наборы данных из-за отсутствия данных.
- Невозможно объединить данные этих веб-сайтов. Необходимо создать три отдельные рекомендательные системы.
- Вы можете объединить три набора данных в один, но сначала вы должны нормализовать каждый набор данных отдельно, вычтя среднее значение, а затем разделив на (max - min), где max и min (5-1) или (10-1) или (100-1) для трех веб-сайтов соответственно.

4. Что такое рекомендательная система, основанная на содержании?

- Рекомендательная система, основанная на контенте, пытается рекомендовать товары пользователям на основе их профиля, основанного на их предпочтениях и вкусах.
- Рекомендательная система, основанная на содержании, пытается рекомендовать элементы на основе сходства между пунктами
- Все вышеперечисленное.

5. Что такое рекомендательная система на основе памяти?

- В подходе, основанном на памяти, рекомендательная система создается с использованием методов машинного обучения, таких как регрессия, кластеризация, классификация и т. д.
- В подходе, основанном на памяти, разрабатывается модель пользователей в попытке узнать их предпочтения
- В подходе, основанном на памяти, мы используем весь набор данных пользовательских элементов для создания системы рекомендаций.

6. Что означает «холодный старт» в коллаборативной фильтрации?

- Сложность рекомендаций, когда у нас недостаточно оценок в наборе данных пользовательского элемента
- Сложность в рекомендации, когда у нас есть новый пользователь, и мы не можем сделать для него профиль, или когда у нас есть новый элемент, который еще не получил никакого рейтинга.
- Сложность рекомендаций, когда количество пользователей или элементов увеличивается, а объем данных расходуется, поэтому алгоритмы начнут страдать от падения производительности.

Домашняя работа

Примерный перечень тем

1. Разработка рекомендательной системы на основе коллаборативной фильтрации.
2. Проектирование алгоритма под рекомендательную систему.
3. Оценка рекомендательных систем.

Примерные задания

1. Разработать рекомендательную систему на основе коллаборативной фильтрации. Выполнить заданий с использованием Python, Pandas, SKLearn, MatplotLib, SeaBorn, Jupyter Notebook.
2. Источник кода с комментариями : http://blog.ethanrosenthal.com/2016/01/09/explicit-matrix-factorization-sgd-als/
- Сравнить качество рекомендаций и время выполнения ALS (без базовых предикторов - without biases) и SGD (без базовых предикторов) для различных параметров регуляризации (например, 0,001, 0,01,...) и числа факторов (например, 1, 10, 25, 50,...). Для SGD необходимо усовершенствовать код.
- Исправить код ALS так, чтобы была возможность производить расчеты с базовыми предикторами (biases). Неплохая инструкция тут: http://activisiongamescience.github.io/2016/01/11/Implicit-Recommender-Systems-Biased-Matrix-Factorization/.
- Сравнить все 4 подхода по мере MSE, времени на обучение (training phase) и тестирование (inference phase). Результаты, помимо шагов выполнения в ipython, необходимо представить в сводной таблице.

Источник данных: MovieLens 100k. Может быть выбран альтернативный, но не меньший по числу оценок, либо пользователей и предметов (items).

3. Сравнение методов коллаборативной фильтрации по сходству пользователей и по сходству объектов

1. Требуется реализовать вычисление ошибки [MAE](http://www.recsyswiki.com/wiki/Mean_absolute_error) и [RMSE](http://www.recsyswiki.com/wiki/Root_mean_square_error-mean-square_deviation) на тестовых данных [Movie Lens](http://grouplens.org/datasets/movielens/).

В качестве данных обучения можно использовать файлы с расширением base, а тестирование качества провести на файле test: пары файлов u1.base и u1.test, ..., u5.base и u5.test. Каждая пара — это разбиение 80%/20% данных для всех пользователей u на обучающие и тестовые данные.

2. Для каждого метода (user-based и item-based) постройте графики зависимости [MAE](http://www.recsyswiki.com/wiki/Mean_absolute_error) и [RMSE](http://www.recsyswiki.com/wiki/Root_mean_square_error-mean-square_deviation) от числа соседей (диапазон от 1 до 100 с разумным шагом).
3. Если качество предсказаний слишком низкое (MAE>2,0), то попробуйте формулы 2.6 и 2.7 из обзора http://files.grouplens.org/papers/FnT%20CF%20Recsys%20Survey.pdf.

Можно использовать альтернативные формулы для исходной модели $r_{u,i} = k\sum\limits_{u^\prime \in U}\operatorname{sim}(u,u^\prime)r_{u^\prime, i} \mbox{ (случай user-based модели):}$

$$r_{u,i} = \frac{1}{N}\sum\limits_{u^\prime \in U}r_{u^\prime, i}$$

$$r_{u,i} = \bar{r_u} + k\sum\limits_{u^\prime \in U} sim(u,u^\prime)(r_{u^\prime, i}-\bar{r_{u^\prime}}) \mbox{, где } k =1/\sum_{u^\prime \in U}|\operatorname{sim}(u,u^\prime)|.$$

4. Сравните подходы на основе полученных результатов по аналогии с пунктами 1 и 2.

5. Как изменяется величина MAE (RMSE) от числа выдаваемых рекомендаций (top-n): $n \in \{1,3,5,10,15,20,30,40,50,100\}$?
6. Как Вы считаете, какие фильмы чаще рекомендуются -- популярные с высокими оценками или редкие (те, которые редко оцениваются) с высокими оценками?
7. Что делать, если соседей (то есть похожих на целевого пользователя или конкретный товар) мало? Нужно/можно ли как-то учитывать достоверность таких рекомендаций?

Экзамен

Список примерных вопросов

1. Типы рекомендательных систем
    1) Фильтрация на основе контента
    2) Совместная фильтрация
2. Рекомендательные системы на основе контента
    1)Анализ документов с помощью TI-IDF
    2)Создание векторизатора TF-IDF
    3)Вычисление косинусного подобия – скалярного произведения нормализованных векторов
    4)Преимущества рекомендательных систем, основанных на контенте
    5)Ограничения рекомендательных систем, основанных на контенте
3. Рекомендательные системы совместной фильтрации
    1) Совместная фильтрация на основе памяти
    2) Рекомендательная система совместной фильтрации на основе моделей
4. Оценка рекомендательной системы

Список литературы

Электронные ресурсы (издания)

1. Isinkaye F.O., Folajimi Y.O., Ojokoh B.A.: «Recommendation systems: Principles, methods and evaluation», 2015 г. - [Электронный ресурс]: https://www.sciencedirect.com/science/article/pii/S1110866515000341
2. Гастон В. «Introduction to Recommender Systems in 2018», 2018 г. - [Электронный ресурс]: www.tryolabs.com/blog/introduction-torecommender-systems
3. Гончаров М. «Data mining: рекомендательные системы. Collaboration Filtering», 2012 г.
4. А. Константинов «Рекомендательные системы: тематический обзор» // Национальный Открытый Университет «ИНТУИТ», 2012- [Электронный ресурс]: www.hse.ru/data/2012/12/20/1303750691/Block-3.pdf

Печатные издания

1. Kamyshev V K., Kureichik V.M., Borodyanskiy I.M., “Review of the Recommender Systems Application in Cardiology”, Cardiometry, 2020, no. 16, 97–105
6.И. Ю. Квятковская, Чанг Во Тхи Хуен, Тоан Чан Куок, “Модель и алгоритм поддержки принятия решения по выбору продуктов для рекомендации пользователю на основе метода анализа статистической импликации”, Вестн. Астрахан. гос. техн. ун-та. Сер. управление, вычисл. техн. информ., 2023, № 2, 116–124