РЕКОМЕНДУЮЩИЕ СИСТЕМЫ

Гниломедов Иван, гр. 3538
Васильева Екатерина, гр. 3539

Оглавление:

1. ПРЕДИСЛОВИЕ

Рассмотрим такую ситуацию: в вашем загруженном графике появилась свободное время, и вы решили посмотреть какой-нибудь интересный фильм (почитать интересную книгу или послушать приятную музыку). Вот в этот момент вам неизбежно придется решать проблему выбора: какой же конкретный фильм посмотреть? Современный шоубизнес готов предложить нам невероятно огромное количество фильмов, музыки и т. д. Очевидно, что ни один человек не в состоянии пересмотреть всех снятых фильмов. А даже если кто-нибудь и решится на подобный подвиг, то, несомненно, за то время, пока он будет этим заниматься, режиссеры успеют наснимать ещё...

Однако не все фильмы придутся вам по вкусу, а не посмотрев фильм, сложно сказать понравится он вам, или нет. Вот именно эту проблему и призваны решать рекомендующие системы.

Итак, *Рекомендующие системы* (Recommender systems) работают с информацией, полученной после обработки истории предпочтений пользователей. Наиболее распространенные подходы построить рекомендующую систему – *общая фильтрация* (Collaborative Filtering) и *частичная фильтрация* (Content-based recommending).

Общая фильтрация систем рассматривает взгляды пользователей (часто в форме голосования) на интересующие вещи. После обработки этой информации она сравнивает профили пользователей с целью рекомендовать им ту, или иную вещь. Методы, основанные на *частичной обработке*, дают рекомендации после сравнения информации о предмете рекомендации с представлениями о том, что интересно пользователю.

2. НЕМНОГО ИСТОРИИ

Общая фильтрация имеет свои корни в самых ранних системах фильтрации информации, где по запросу предоставляется информация после обработки поведения пользователей. Она была неэффективна тем, что не могла помочь изучению *web.* Так же к её недостаткам можно отнести и необходимость достаточно больших временных затрат на построение первичной базы данных, по которой она могла бы делать эффективные рекомендации.

Первая система, использующая общую фильтрацию, была *Information Tapestry project at Xerox PARC*. Эта система разрешала пользователям искать документы, учитывая замечания пользователей, которые были сделаны до этого. Было много проблем с этой системой, так как она работала лишь для небольшой группы людей и должна была отвечать на вопросы с различной лексикой, что очень проигрывало самому замыслу общей фильтрации.

3. ПРИМЕНЕНИЕ

Системы рекомендаций имеют различное использование в наши дни. На пример, системы рекомендаций пользователям различных вещей. В том числе книг, фильмов, дисков, страниц в Интернете, новостей и т.д. Многие он-лайн магазины снабжены подобными рекомендующими сервисами, на пример, Amazon, CDNOW, BarnesAnd-Noble, IMDb и другие. Их использование помогает повысить продажи.

USENET[1] использует принцип общей фильтрации. Она доступна многим пользователям и использует простой способ доступа к статьям. Система позволяет оценивать материал, что впоследствии другим пользователям дает возможность поиска статей, основываясь на этих оценках.

[1] Usenet - система телеконференций Internet. Организована как большой (содержащий более 12 тысяч конференций) иерархический каталог, узлами которого являются группы новостей по определённым предметным областям. Сообщения, присылаемые пользователями, обычно не задерживаются в сети больше пяти дней

4. ПОСТАНОВКА ЗАДАЧИ

Рассмотрим N наименований какой-либо мультимедиа информации (фильмов, саундтреков, книг и пр.) I1, I2, ... IN и M пользователей U1, U2, ... Um. Каждый пользователь предоставляет список понравившихся ему наименований. На основании этой информации рекомендующая система каждому пользователю должна выдать список наименований, которые потенциально могут ему понравиться и, по возможности, оценить с какой вероятностью они придутся пользователю по вкусу.

Можно так же рассматривать ситуацию, когда пользователь предоставляет и список того, что ему категорически не понравилось. Или, даже, оценивает каждый образец по какой-либо шкале (например, от -10 до +10).

5. ФУНКЦИОНАЛЬНОСТЬ РЕКОМЕНДУЮЩИХ СИСТЕМ

Рекомендующая система должна:

1.Выдавать пользователю список образцов, которые ему, вероятно, понравятся. При этом пользователь предоставляет список образцов, которые ему нравятся и после этого система анализирует свою внутреннюю базу.

2.Осуществлять сбор и хранение данных о предпочтениях пользователей.

3.В процессе работы производить самообучение (автоматически регулировать свои внутренние параметры, по которым она оценивает вероятность того, что данный образец понравится данному пользователю).

6. АЛГОРИТМЫ РЕКОМЕНДАЦИЙ

Самый простой алгоритм, который приходит в голову – это подсчёт частоты встречаемости каждого наименования. Каждому пользователю, при этом, рекомендуются наиболее популярные образцы. Это один из немногих случаев, когда самый простой способ не является самым правильным. Во-первых, заметим, что предпочтения одного конкретного пользователя не оказывают ощутимого влияния на общий рейтинг (мы предполагаем, что пользователей достаточно много). В этом случае всем людям мы рекомендуем одно и то же не зависимо от предоставленного ими списка. А интересы у каждого разные...

Придумаем что-нибудь лучше. Любой алгоритм основывается на своих структурах данных. Возможны два способа их хранить: помнить полностью исходные списки понравившихся пользователям образцов, или хранить некоторые обобщённые данные о предпочтениях всех пользователей сразу, и при поступлении новой информации обновлять всю базу.

6.1 ОЦЕНКА ПОХОЖЕСТИ ПОЛЬЗОВАТЕЛЕЙ

В первом случае алгоритм должен вести себя следующим образом: при поступлении очередного запроса (списка понравившихся наименований) мы пробегаем по всей базе данных и оцениваем степень похожести текущего пользователя из Б.Д. на пользователя, от которого поступил запрос. Похожим людям, вероятно, должны нравиться похожие образцы. Следовательно, у нас есть возможность сформировать ожидаемый рейтинглист для данного человека на основании рейтинглистов похожих на него пользователей. Далее мы исключаем из этого рейтинга уже известные образцы (предоставленные самим пользователем) и выдаём ему оставшийся список.

Неопределенным остался только способ оценки похожести пользователей. Единственное, на что мы можем опираться – это пересечение множеств понравившихся объектов и/или на то, насколько близкие оценки дали им сами пользователи. Кроме того, можно предложить следующую эвристику: найдём общий рейтинг каждого образца (то, насколько он популярен среди вообще всех пользователей) и совпадению мнений по более редкому образцу будем приписывать больший вес.

В полезности такого подхода можно убедиться, рассмотрев следующий пример. Среди множества исполнителей выделим 2, отличающихся по стилю. Пусть первый из них выпустил 5000 песен (он очень популярен среди масс), а второй всего 50 (новичок). Рассмотрим 3-х

пользователей: U1 нравятся оба, а U2 нравится второй, но не нравится первый, а U3 - наоборот, нравится первый, но не нравится второй. U1 похож на U2 и на U3 в равной степени, но при оценке с равными весами, 50 будет всего 1% от 5000 и наша старая система скажет, что U1 и U3 очень похожи, а U1 и U2 не похожи вовсе.

6.2 ОЦЕНКА ПОХОЖЕСТИ ОБРАЗЦОВ

В качестве альтернативного способа можно хранить матрицу так называемого item to item соотношения, т. е. степень похожести одного образца на другой.

В этом случае мы исходим из предположения, что если пользователю Pk нравится Ig, а Ig очень похож на Ih то, вероятно, пользователю Pk понравится и Ih. Интуитивно вполне правдоподобное предположение.

Однако в этой ситуации вырисовывается следующая проблема: как было упомянуто ранее количество наименований достаточно велико, а хранение подобной item to item матрицы потребовало ба квадратичной памяти... а где её столько взять? Выход один: нам придётся хранить не всю матрицу. Как же с этим справиться? Исходя из предположения, что каждый образец вряд ли сильно похож на все остальные, мы можем сделать вывод, что матрица будет весьма разреженной. Тогда достаточно для каждого наименования хранить список похожих на него, а ещё лучше не весь список, а только самые похожие элементы. В этом случае мы, конечно, решим проблему размера базы, но получим ещё один нежелательный эффект.

Рассмотрим нашу БД в какой-либо момент времени. Пусть нам поступила информация для обновления, содержащая новый элемент (которого ещё не было в нашем топ листе). Но тогда этому элементу будет приписана минимальная степень похожести. Естественно, он не сможет конкурировать с теми, которые были там уже давно (неудачник) и сразу будет пущен в расход. Получается, что наша система будет стремиться к какому-то стабильному, самовоспроизводящемуся состоянию и, в конце концов, её развитие остановится. Мы этого допустить не можем. Что же делать?

А пусть для каждого наименования у нас будут храниться два списка: топ лист - наиболее похожие по итогам всей истории развития, и список «новобранцев» - наиболее похожие по итогам нескольких последних запросов. Новобранец живёт в этом списке какое-то время и потом, если он оправдал оказанное ему доверие и приобрёл большую степень похожести, то он переходит в список для "дедушек", иначе – удаляется. Чтобы список новичков не разрастался, сделаем его размер ограниченным сверху. То, кого следует удалить при переполнении, определяется, например, по splay стратегии.

Замечание: по аналогии с предыдущим пунктом, пользователю, который указал мало наименований, можно приписывать больший вес.

Особенностью такого метода является то, что происходит кластеризация образцов. Интуитивно этот эффект является оправданным, так как мы ищем степень похожести объектов, как и при кластеризации.

6.3 КОМБИНИРОВАННЫЙ АНАЛИЗ

Как принято говорить на наших семинарах, помимо всех перечисленных методов, на практике часто используется и комбинированный принцип. Но, вопреки ожиданиям, мы два метода, обсуждавшиеся выше, будем использовать не параллельно, а последовательно: сначала оценим похожесть образцов, а затем будем рассчитывать похожесть пользователей не просто по пересекающемуся множеству любимых фильмов, книг и т. д. а по схожести так называемого псевдорейтинга.

Как уже упоминалось ранее, матрица отношения *user to item* очень разрежена, то есть пользователь не может пересмотреть почти все фильмы и иметь о них своё мнение. Но при оценке похожести пользователей желательно, чтобы были заполнены все ячейки. Действительно, в незаполненные ячейки можно вставить предварительные значения рейтингов, рассчитанных

исходя из анализа матрицы *item to item*, в результате чего будет получен упомянутый псевдорейтинг.

$$R\_pseudo_{a,i} = \sum_{j=1}^{n} r_{a,j} * Same(i,j) \qquad (1)$$

где $r_{a,i}$ рейтинг образца *i,* предоставленный пользователем *a*, $\bar{r}_a$ – среднее значение рейтингов пользователя.

Далее пользуемся псевдо рейтингом.

Применим *neighborhood-based algorithm*. В качестве меры различия будем использовать так называемую корреляционную (она же косинусная) меру, которая вычисляется по формуле:

$$Sim_{a,b} = \frac{\sum_{i=1}^{m} (r_{a,i} - \bar{r}_a) * (r_{b,i} - \bar{r}_b)}{\sqrt{\sum_{i=1}^{m} (r_{a,i} - \bar{r}_a)^2 * \sum_{i=1}^{m} (r_{b,i} - \bar{r}_b)^2}} \qquad (2)$$

где $Sim_{a,u}$ - похожесть *a* и *u*.

Выделим *k* ближайших соседей, и для вычисления итогового рейтинга каждого образца воспользуемся формулой:

$$P_{a,i} = \bar{r}_a + \frac{\sum_{u=1}^{k} (r_{u,i} - \bar{r}_u) * Sim_{a,u}}{\sum_{u=1}^{k} (r_{u,i} - \bar{r}_u)} \qquad (3)$$

где $P_{a,i}$ - ответ на вопрос «насколько пользователю *a* понравится образец *i*».

Замечание: внимательно посмотрев на формулу, можно убедиться, что сложность расчетов составляет *n* * *m* (число пользователей на число образцов), что нас не устраивает. Опять воспользуемся разреженностью: псевдорейтинг тоже, вероятно, обладает этим свойством.

Такой подход кажется концептуальным. Более того, во многих книгах, статьях, посвященных рекомендующим системам, в разделе *experiments* именно ему приписывают наибольшую точность.

7. СБОР И ХРАНЕНИЕ ДАННЫХ

Наиболее простой и эффективный способ получать данные о предпочтениях пользователей – это просто запоминать все их запросы. Действительно, та информация, которую пользователь вводит при попытке получить рекомендацию, и является рабочим материалом при формировании собственно БД.

Это вполне удобно для пользователя в том случае, если он ищет какую-нибудь крупную информацию, например, фильм или книгу (то, что займёт его надолго). Но не всё так просто, если вы хотите найти музыку. Во-первых, вам выдадут огромный список того, что вам потенциально может понравиться, и, во-вторых, вам придётся вводить длинный список всего того, что вам нравится. Хотелось бы, чтобы всё это происходило как-то автоматически, лучше даже, чтобы пользователь вовсе не тратил никаких усилий при получении рекомендаций.

Хотелось? - получите! Например, можно встроить шпиона в *Winamp* (спросив при этом самого пользователя конечно 8-)), который бы следил за тем, какие саундтреки пользователь ставит чаще всего, что он не прочь прослушать повторно. Далее эти данные отправляются на сервер (опять же, с разрешения самого пользователя) и приходит ответ: «Дорогой пользователь, тут рекомендующая система посоветовалась и решила, что вам, наверняка, придутся по вкусу следующие треки: ...» . Вот так, можно сказать, бесплатно мы получили ценный совет и пополнили базу данных.

8. УЧИМСЯ НА ОШИБКАХ

При конструировании алгоритмов мы оставили кое-что недоопределённым. В любом случае, при анализе степени схожести (пользователей или образцов) нам приходится подбирать какие-то коэффициенты при складывании разных параметров одинаковости в один общий индекс. А вполне может случиться так, что оптимальный принцип зависимости между промежуточными вычислениями и финальным результатом и вовсе не линеен. В конце концов, это можно даже доверить такому модному принципу как нейронные сети.

В любом случае, нам нужен механизм, который задал задачу рекомендательной системе, взглянул на результат и сказал: "Точность предсказания оставляет желать лучшего. Коммунизм с такими рекомендациями не построить". Рекомендующая система взвесит все «за» и «против» и выдаст нам ответ лучше. Ну и так далее.

Как же организовать автоматическую систему проверки качества предсказания? Можно, конечно, попробовать попросить пользователя потом сказать нам, насколько наша рекомендация была точна. Но стоит ли утруждать занятого человека, если можно справиться самим?

Будем действовать по следующему образцу. Пользователь предоставил нам список его любимых мелодий, а мы возьмем и выкинем из него 1 / 4, а затем запустим нашего рекомендателя. Далее просто посмотрим на то, осилил ли рекомендатель предсказать выкинутую часть. Вот теперь мы можем выдать нашей системе правильный ответ, и пусть она делает работу над ошибками. Так можно повторить несколько раз (но не стоит увлекаться - всегда есть опасность научить нашу систему хорошо работать только с одним пользователем). Так мы убиваем сразу двух зайцев - обучим систему и сможем сказать пользователю, насколько точен наш прогноз. Клёво да?

9. РЕЗУЛЬТАТЫ

Как видно из способа организации рекомендующих систем, подобный механизм пригоден не только для предоставления рекомендаций конкретному потребителю, но и, например, для получения рейтингов фильмов или саундтреков. Более того, этот механизм позволяет более тонко проводить маркетинговые исследования. Действительно, так как мы производим кластеризацию данных, то мы можем делать выводы о популярности того или иного произведения не только для толпы в целом (что иногда бессмысленно), но и оценивать насколько данная песня или данный фильм приветствуется в разных слоях населения.

10. ССЫЛКИ

- "Distributed Collaborative Filtering for PeertoPeer File Sharing Systems"
  (Jun Wang, Johan Pouwelse).
- "Unifying Userbased and Itembased Collaborative Filtering Approaches by Similarity Fusion"
  (Jun Wang, Arjen P. de Vries, Marcel J.T. Reinders).
- "Content-Boosted Collaborative Filtering for Improved Recommendations"
  (Prem Melville, Raymond J. Mooney, Ramadass Nagarajan: Department of Computer Sciences University of Texas).
- "Recency-Based Collaborative Filtering"
  (Yi Ding, Xue Li, Maria E. Orlowska).