# Архитектура рекомендательной системы, работающей на основе неявных пользовательских оценок

© А.Н. Федоровский, В.К. Логачева

Mail.Ru
fedorovsky@corp.mail.ru, v.logacheva@corp.mail.ru

**Аннотация**

Рекомендательные системы – сравнительно новый класс ПО, в чью задачу входит изучение вкусов пользователя путем анализа его действий и оценок. В работе описано построение такой системы. Упор сделан на скорость работы, устойчивость результатов, работу с неявными пользовательскими оценками (implicit datasets) и с разными наборами данных. Детально расписана методика работы и оптимизируемые меры качества.

## 1. Введение

Рекомендательная система позволяет человеку обозначить свои вкусы и возвращает результаты, любопытные для него, базируясь на оценках других пользователей и своих предположениях.

В отличие от поисковых систем, для получения от системы ответа не требуется четкого задания запроса. Вместо этого пользователю предлагается оценить некоторые объекты из коллекции, и на основании этих его оценок и сравнения их с оценками предыдущих пользователей строятся предположения о вкусах пользователя и возвращаются наиболее близкие к ним результаты, формируя для него персонализированную выдачу.

В качестве набора оцениваемых объектов могут, к примеру, выступать: каталог ссылок на веб-сайты, лента новостей, товары в электронном магазине, коллекция книг в библиотеке и т.п.

В сферу применения подобных систем входят и ситуации, когда пользователь не ищет информацию по конкретному ключевому слову, а, к примеру, хочет получить список современных статей, похожих по тематике на те, которые он просматривал до этого.

Рекомендательными системами занимаются с начала 90-х годов [1,5], однако серьезнейшим толчком к развитию этой темы явился конкурс Netflix Prize [2], организованный в 2006 г. компанией Netflix, одним из лидеров на американском рынке проката DVD. Компания предоставила участникам данные о 100 млн. рейтингов, проставленных 480 тысячами пользователей 18 тысячам фильмов в течение 6 лет. Участник, показавший результаты на 10% лучше, чем их собственный движок рекомендаций, получал миллион долларов. И набор данных, и приз в десятки раз превышали аналоги. В итоге методы построения рекомендательных систем начали бурно развиваться, но задача оказалась сложной – несмотря на ажиотаж и жесточайшую конкуренцию, на выполнение условий у участников ушло три года.

Среди подходов выделяются три:

1) Content-based [10] – строятся профили пользователей и объектов на основе анализа текстовой метаинформации объектов. Затем, с помощью некой меры близости, часто – коэффициента Пирсона, определяются объекты, близкие к профилю пользователя. К сожалению, несмотря на богатое наследие Information Retrieval, этот метод пока не удалось заставить работать эффективно. Например, в работе [6] строится система рекомендации интересных твитов в Twitter и явно видно, что вклад в качество системы от content-based компоненты сильно ниже, чем от коллаборативной фильтрации (CF), к которой принадлежат два следующих подхода:

2) Neighbourhood-based (NB) [8,12] – подходящими объектами для пользователя $u$ считаются объекты, высоко оцениваемые его «соседями», т.е. такими пользователями $v_i$, у которых оценки хорошо скоррелированы с $u$.

3) Model-based, matrix factorization (MF) [8,12,13] – предполагается, что оценка пользователя определяется не очень большим числом (K=50-200) скрытых факторов (например, фильм американский или азиатский; блокбастер или арт-хаус; старый или современный и т.п). И задача здесь состоит в том, чтобы найти такие факторы, при выборе которых эта приближенная модель будет лучше всего приближать реальные пользовательские рейтинги. Для этого часто используется сингулярное разложение матриц [8] или итеративное градиентное приближение [4,12,13]. Один их таких алгоритмов будет подробно рассмотрен ниже.

Считается, что MF лучше справляется с нахождением закономерностей в рамках коллекции

**Труды 13[й] Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL'2011, Воронеж, Россия, 2011.**

в целом, и, будучи примененным в одиночку, дает более высокое качество, чем NB. Тем не менее, последний метод лучше находит локализованные особенности. К тому же, эти два метода хорошо сочетаются: [8,12].

## 2. Постановка задачи

Выбрать и реализовать алгоритм, который бы

- легко масштабировался до значительно большего, чем Netflix, размера
- работал быстро
- адаптировался бы к разным наборам данных
- работал бы как с явными рейтингами (оценки), так и с неявными (просмотры)
- позволял бы на лету добавлять новые рейтинги, пользователей, объекты.

В качестве базы для старта был выбран подход MF, а конкретно алгоритм BRISMF, полностью описанный в [13].

### 2.1. Идея BRISMF

Пусть даны пользователи $1 \dots N$ и объекты $1 \dots M$. Построим частично определенную разреженную матрицу рейтингов $R \in \mathbb{R}^{N \times M} = (r_{ui})$, где $r_{ui}$– оценка пользователем $u$ объекта $i$. Нужно найти такое разложение

$$R \approx \hat{R} = PQ,$$

где $P \in \mathbb{R}^{N \times K}$ – матрица профилей пользователей, сопоставляющая каждому пользователю $u$ вектор-строку $p_u \in \mathbb{R}^K$, $Q \in \mathbb{R}^{K \times M}$ – матрица профилей объектов, сопоставляющая каждому объекту $i$ вектор-столбец $q_i \in \mathbb{R}^K$, а $K$ – выбранное число скрытых признаков; чтобы минимизировать ошибку

$$RMSE = \sqrt{\sum_{r_{ui} \in R} \frac{(\hat{r}_{ui} - r_{ui})^2}{|\{r_{ui} \in R\}|}}$$

Поскольку

$$\hat{r}_{ui} = p_u q_i = \sum_{k=1}^{K} p_{uk} q_{ki},$$

RMSE как функция от $p_{uk}, q_{ki}$ минимизируется методом градиентного спуска. Это сводится к коррекции векторов $p_u, q_i$ для каждого рейтинга $r_{ui}$:

$$\forall k \;\; p'_{uk} = p_{uk} + з_p e_{ui} q_{ki}$$
$$\forall k \;\; q'_{ki} = q_{ki} + з_q e_{ui} p_{uk}$$

где $e_{ui} = \hat{r}_{ui} - r_{ui}$ – вектор ошибки, а з – параметры скорости обучения.

Для контроля переобучения нужно добавить регуляризационные параметры л:

$$\forall k \;\; p'_{uk} = p_{uk} + з_p(e_{ui} q_{ki} - л_p p_{uk})$$
$$\forall k \;\; q'_{ki} = q_{ki} + з_q(e_{ui} p_{uk} - л_q q_{ki})$$

Также, перед запуском минимизации, для уменьшения разброса оценок, возникающего из-за того, что какие-то объекты лучше других и, значит, всеми оцениваются выше, есть смысл вычесть индивидуальные смещения рейтингов для каждого пользователя и объекта:

1) Из всех элементов матрицы $R = (r_{ui})$ вычитаем среднее $R_{\text{сред}} = \frac{\sum_{r_{ui} \in R} r_{ui}}{|\{r_{ui} \in R\}|}$
2) Из каждой строки $r_u$ матрицы $R = (r_{ui})$ вычитаем среднее $r_{u\,\text{сред}} = \frac{\sum_{r_{ui} \in r_u} r_{ui}}{|\{r_{ui} \in r_u\}|}$
3) Из каждого столбца $r_i$ матрицы $R = (r_{ui})$ вычитаем среднее $r_{i\,\text{сред}} = \frac{\sum_{r_{ui} \in r_i} r_{ui}}{|\{r_{ui} \in r_i\}|}$

В исходном описании BRISMF величины смещений были такими же настраиваемыми параметрами, как и элементы P и Q. При реализации такого варианта была замечена неустойчивость в величинах смещений, а, поскольку они влияют на поведение системы в целом сильнее, чем индивидуальные компоненты векторов профилей, было решено вместо настраиваемых смещений использовать в точности среднюю оценку. Было также отмечено незначительное повышение качества работы системы – RMSE при прочих равных уменьшился на 0.0002. Кроме того, средние оценки полезны сами по себе, так как $R_{\text{сред}} + r_{u\,\text{сред}} + r_{i\,\text{сред}}$ можно рассматривать как начальное приближение для рекомендаций.

## 3. Неявные данные

Массив данных Netflix, имея беспрецедентный размер, содержал явные рейтинги (explicit dataset), собрать которые от пользователей иногда представляется затруднительным. Часто удобнее и быстрее собрать не осознанную оценку объектов от пользователя, а проанализировать лог его действий, оценивая, к примеру, просмотр страницы с описанием объекта как неявную положительную оценку этого объекта. Отсутствие просмотра в этом случае говорит об отсутствии интереса и может считаться отрицательной оценкой. Такой набор данных называется неявным (implicit dataset).

В случае явных рейтингов и положительные, и отрицательные оценки пользователей сконцентрированы в малой части матрицы рейтингов, она сильно разрежена. Если же каждое отсутствие просмотра страницы объекта считать за отрицательную оценку, то придется обрабатывать не малую часть матрицы рейтингов, а всю ее - кардинально возрастает сложность задачи. И, вдобавок, не разделяются объекты, действительно неинтересные пользователю и те, которые он просто еще не просматривал.

Для работы с неявными данными были разработаны специальные разновидности явных алгоритмов. Например, IALS1 [11] – более быстрая версия IALS [7] или сэмплинг из непросмотренных данных с бэггингом [9].

Их слабая сторона заключается в скорости работы – IALS1, быстрейший из них, работает в десятки раз медленнее RISMF. Поэтому было решено сформировать синтетические явные рейтинги на основании логов различных действий пользователя:

$$Rate = f_0\left(\sum w_i f_i(c_i)\right)$$

где $c_i$ – количество i-х действий, совершенных пользователем с объектом, $w_i$ – вес действия, а $f()$ – функции с насыщением. Например, для коллекции приложений из социальной сети мы рассматривали следующие действия: установка, удаление, использование, платежи, нажатие на кнопку «нравится», а также явные оценки пользователей.

К таким наборам искусственных оценок уже можно применить тот же алгоритм BRISMF, что и в случае явного набора данных.

На некоторых наборах данных матрица оценок получалась сильно разреженной (плотность 0.01%-0.1% против 1.16% у Netflix). В таких случаях, для выявления более плотного набора данных, производился отбор пользователей и объектов с большим числом оценок. Если же подавляющее большинство оценок пользователей одинаково, например, при наличии в наборе данных только одного вида действий, которое, к тому же, обычно выполняется пользователем лишь единожды (таковы, например, добавление веб-страницы в список избранных, установление дружбы между пользователями и т.п.), может помочь «естественная» кластеризация объектов. Характер ее зависит от предметной области: если объекты – ссылки на страницы сайтов, их можно сгруппировать по сайтам, если музыкальные композиции – по авторам, новости – по сюжетам. За счет некоторого огрубления точности происходит многократный рост числа нетривиальных значений оценок, что позволит лучше обучиться и, в итоге, поднять качество работы системы.

## 4. Архитектура системы

В качестве входных данных система получает информацию о действиях, совершенных пользователем, или об оценках, выставленных им объектам из набора данных. Поскольку данные предполагается как получать, так и использовать на веб-проектах, основной внешний интерфейс был реализован поверх протокола HTTP, по которому в формате JSON передаются как действия и оценки пользователя, так и запросы рекомендаций для него. Эти запросы могут быть совершены как с помощью AJAX из Javascript-кода, размещаемого на страницах, так и, при желании, с сервера проекта, пользующегося услугами рекомендательной системы. Такой подход позволяет начать использование рекомендательной системы на новом проекте с минимальными усилиями, сравнимыми с установкой на проект счетчика.

Архитектурная схема модулей проекта представлена на рисунке 1. Здесь сплошные стрелки обозначают получение данных, а пунктирные – запросы рекомендаций.

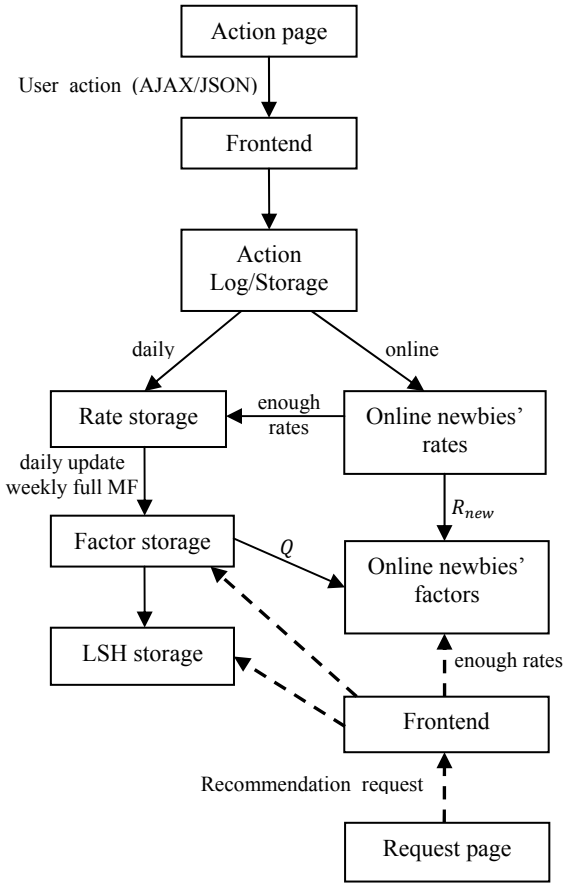


Рис. 1. Модули рекомендательной системы.

Информация о действии, совершенном пользователем, попадает на фронтенд-сервер рекомендательной системы, а оттуда – в модуль хранилища действий (Action Log/Storage), совмещающий, как видно из названия, неупорядоченный лог действий и агрегирующее хранилище.

Время от времени, когда накапливается достаточно много новых действий, для тех пар пользователь-объект, к которым они относились, на основе лога новых и предыдущих действий вычисляются явные оценки и сохраняются в хранилище оценок (Rate storage). Это касается только тех пользователей, которые уже есть в системе, работа с новыми пользователями будет описана ниже.

Для старых пользователей также перерассчитываются вектора профилей в пространстве скрытых признаков и сохраняются в Factor storage. Профили объектов при этом не изменяются. Достаточно редко идет полный пересчет матриц профилей пользователей и объектов. Для полного расчета используется только ядро оценок – выборка из пользователей и объектов, имеющих достаточно большое число оценок и образующих достаточно плотную матрицу. На основании полученных факторов строятся профили для менее активных пользователей и менее популярных объектов.

Подобную схему используют и модули Online newbies' rates и Online newbies' factors, организующие расчет оценок и профилей на лету для новых пользователей и объектов, а также для старых пользователей, только что внесших в систему свои новые действия и оценки. Их профили рассчитываются на основе хранящихся в Factor storage достаточно стабильных профилей объектов. Наличие этих двух модулей позволяет выдавать рекомендации для пользователей с учетом их последних оценок без задержки на перерасчет матриц профилей P и Q.

Оценки новых пользователей по достижении ими достаточно большого количества совершенных действий сохраняются в Rate storage.

Запрос рекомендаций сначала порождает запрос профиля пользователя в Online newbies' factors, а, при его отсутствии там, в Factor storage. Рекомендации для пользователей, совершивших пренебрежимо мало действий (к примеру, меньше 10), системой не возвращаются – их профили пока что построены недостаточно хорошо и рекомендации им лучше строить без учета CF. Например, если важна удовлетворенность пользователя во время холодного старта, можно возвращать самые популярные или самые высокооцениваемые объекты. Или же, с другой стороны, для ускорения обучения есть смысл показывать объекты, наиболее характерные для конкретных факторов или наиболее противоречивые – с максимальным разбросом оценок пользователей.

Для формирования рекомендаций при запросе пользователя в идеальном случае нужно перебрать все объекты в системе, отсортировать их определенным образом и выбрать несколько лучших. В следующем разделе будут описаны как методы сортировки и ограничения числа просматриваемых объектов, так и необязательный, но важный для скорости работы системы модуль пространственной кластеризации объектов и пользователей, в качестве которого в нашей системе выступает LSH storage. Его задача – сгруппировать вектора, находящиеся рядом в пространстве скрытых признаков. Это позволяет в случае большого числа объектов ($10^5 - 10^6$ и больше) резко снизить время отбора объектов-кандидатов на попадание в результирующую выдачу за счет некоторого снижения качества.

## 5. Фильтрация и сортировка объектов

В первом варианте формирования выдачи рекомендации для пользователя мы перебирали все объекты в базе, находя те из них, вектора которых близки к вектору пользователя в пространстве скрытых признаков. Был рассмотрен вариант сортировки по предсказанной для пользователя оценке с учетом смещений оценок для пользователя, объекта и всей коллекции

$$Relevance = B + b_u + b_i + \sum_{j=1}^{k} p_{uj} q_{ij}$$

Однако, как было замечено, при такой сортировке рекомендации перенасыщаются объектами с высокой средней оценкой, а выдача теряет как в персонализации, так и в качестве. Поэтому, для поднятия веса объектам, характерным именно для этого пользователя, была испробована сортировка просто по скалярному произведению профилей

$$Relevance = \sum_{j=1}^{k} p_{uj} q_{ij}$$

Эти методы описаны в разделе с результатами как Full-Bias и Full-SP. Стоит заметить, что полный перебор объектов является достаточно медленным при большом их количестве в системе.

Далее, был использован метод случайных проекций из семейства LSH (locality-sensitive hashing random projection) [3] для группировки пользователей и объектов, близких в пространстве скрытых признаков. Идея метода в следующем: в пространстве скрытых признаков $\mathbb{R}^k$, в котором расположены вектора профилей пользователей и объектов, возьмем семейство гиперплоскостей $\{l_1 \dots l_m\}$. Построим m-битную хеш-функцию над векторами из этого пространства:

$$H_m(v) = \left(h_1 \dots h_m : h_i = sign(l_i \cdot v)\right)$$

$h_i$ определяет, с какой стороны от i-й гиперплоскости находится вектор $v$. Таким образом, семейство $\{l_1 \dots l_m\}$ задает разбиение $\mathbb{R}^k$ на сектора, а $H_m(v)$ задает m-битную нумерацию этих секторов пространства. Эта хеш-функция используется для предварительной фильтрации объектов: при формировании выдачи рекомендации сортируются только объекты, вектора профилей которых попали в тот же сектор пространства, что и вектор профиля пользователя, то есть которые имеют то же значение $H_m(v)$. Этот метод с сортировками, аналогичными предыдущему методу, описан в разделе с результатами как LSH1-Bias и LSH1-SP.

Плоскости были выбраны случайными: вектора их нормалей имели многомерное нормальное распределение. В результате существовал шанс, что сектор пространства $\mathbb{R}^k$, содержащий вектор

профиля пользователя, окажется слишком мал и будет содержать слишком мало объектов, чтобы из них можно было сформировать качественную выдачу. Также вектор профиля пользователя мог располагаться неудачно, например, близко к границе сектора. В этом случае часть объектов, близких к нему, оказывалась принадлежащей другим секторам и также не могла оказаться в выдаче. Для решения этих проблем было решено использовать набор из s m-битных хеш-функций $H_m^1(v) \dots H_m^s(v)$, каждая из которых была построена по своему случайному набору гиперплоскостей, а для сортировки в рекомендательной выдаче отбирались только объекты, совпадающие с профилем пользователя по значению хотя бы одной из этих хеш-функций. Качество по сравнению с одной функцией значительно улучшилось. В разделе результатов эта модификация метода с 4 хеш-функциями обозначена как LSH4-Bias и LSH4-SP.

Также был использован метод предварительной фильтрации объектов на основе информации о соседях: для выдачи сортировались только те объекты, которые оценивались выше среднего пользователями, близкими к данному. Близость считалась по совпадению хеша LSH. В разделе результатов он присутствует как UB-Bias и UB-SP.

## 6. Оптимизация параметров

BRISMF обладает рядом настраиваемых параметров. В реализации системы присутствовало 6 ($з_p, з_q, л_p, л_q$, амплитуды начальных значений P и Q). При выборе плохих параметров алгоритм может начать сходиться медленно или неустойчиво. Поскольку целью было разработать систему, не требующую долгого предварительного тюнинга под конкретный набор данных, к системе была добавлена автоматическая настройка параметров.

Это задача многомерной глобальной оптимизации, где оптимизируемая функция –

$$RMSE\left(з_p, з_q, л_p, л_q, P_{max}, Q_{max}\right).$$

Начальные значения P и Q – случайны, поэтому система даже в одной точке ведет себя нестабильно. В результате от градиентных методов пришлось отказаться – значение градиента оказывалось слишком зашумленным. Метод покоординатного спуска был выбран из-за своей наглядности и показал довольно хорошие результаты. Также был опробован метод имитации отжига, но разницы в качестве отмечено не было. Интересным направлением для будущих исследований может стать замена покоординатного спуск на метод Нелдера-Мида.

В каждой точке (с конкретным набором параметров) алгоритм запускается сходиться несколько раз. В результате виден разброс результатов, который учитывается в оптимизируемой мере качества – цель не просто в снижении RMSE всеми силами, а в компромиссе между малой ошибкой и стабильностью результата. При малом разбросе добавляются несколько контрольных запусков в каждой последующей точке. Многократный запуск алгоритма возможен в первую очередь благодаря высокой скорости его работы.

Для дополнительного контроля переобучения, помимо регуляризации в самом алгоритме и выделенного validation set, повторные запуски с теми же параметрами проводятся на разных сэмплах из него.

Также в оптимизируемой мере неявно учитывается скорость схождения алгоритма, так как на этом этапе алгоритм запускается только фиксированное число итераций и поэтому медленная сходимость ухудшает качество.

Проверив сходимость при разном числе признаков K мы также получили, что оптимальные значения параметров мало изменяются при $K = 5{,}10{,}20{,}50$. Это позволяет запускать предварительную оптимизацию параметров с малым K и использовать оптимальные параметры как начальную точку для оптимизации с б**о**льшими значениями K. Это быстрее, чем оптимизировать параметры сразу с рабочим значением K.

## 7. Мера качества

RMSE как мера оценки качества работы систем рекомендаций весьма популярна, возможно, из-за популярности конкурса Netflix Prize, в котором оптимизировалась именно она. Безусловно, она хороша для проверки системы в целом. Но система рекомендаций возвращает пользователю в выдаче лишь несколько объектов, сочтенных наиболее подходящими и пользователь визуально оценивает качество работы системы именно по ним. В то же время, вклад всех объектов в RMSE одинаков. Имеет смысл сделать упор на оптимизацию местоположения в результатах объектов с наивысшей оценкой. Для этой цели используются, к примеру, меры Top-K из [8] или ARP из [11]. Мы используем меру, похожую на ARP:

Если t объектов из тестового сета получили наивысшую реальную оценку пользователя $u$, но при этом стоят в отсортированном списке результатов для пользователя на позициях $1 \leq 層_1 < \dots < n_t$, а всего результатов при этом T, то мерой считаем

$$A\ P1_u = \frac{\sum_{i=1}^{t}(n_i - i)}{t * (T - t)}$$

то есть нормированную сумму разниц в позициях между реальным и идеальным результатом. А мера на всей коллекции усредняется по всем пользователям.

Слагаемые суммы в числителе, очевидно, всегда положительны и достигают нуля только в случае идеальной выдачи. В самом деле, если i-й по счету в выдаче объект из высоко оцененных пользователем стоит на i-й позиции, то все объекты с первого по i-й оценены пользователем наивысшей оценкой, а, значит, выдача с первого по i-й объект идеальна.

Как следствие, выдача идеальна (все t объектов, получивших наивысшую оценку пользователя, стоят на первых t местах) тогда и только тогда, когда все слагаемые суммы в числителе равны нулю, а, значит, и сама мера $ARP1 = 0$. Аналогично, в случае идеально плохой выдачи (все t объектов, получивших наивысшую оценку пользователя, стоят на последних t местах) i-й по счету такой объект стоит на позиции $n_i = T - t + i$, каждое слагаемое суммы в числителе равно $T - t$, а значение меры $ARP1 = 1$.

Эта мера напоминает ARP, с тем отличием, что ARP, по крайней мере, как она описана в [11], на идеальной выдаче имеет значение не равное нулю, а зависящее от размера рассматриваемой выдачи, что достаточно неудобно.

## 8. Наборы данных и результаты

Для контроля успешности реализации алгоритмов была использована RMSE на полном наборе Netflix, характеристики которого описаны в начале статьи. С использованием K=50 скрытых признаков и повторным перестроением пользовательских профилей (см. 3.5 в [13]) было получено значение RMSE = 0.8998, расчет идет 12 минут в один поток. Наиболее близкая реализация алгоритма, для сравнения – BRISMF#1U из [13], на нем достигнуто RMSE = 0.9072, но между ними есть существенные отличия. Во-первых, вместо фиксированного, как в BRISMF#1U, числа эпох при расчете была использована техника early stopping, в том числе и на этапе перерасчета матрицы P. Во-вторых, использованы средние оценки вместо настраиваемых смещений. В-третьих, была проведена оптимизация параметров модели. И, наконец, отличается число скрытых факторов – 50 против 40. Впрочем, оптимизация RMSE не входила в задачу, основное внимание было уделено формированию качественной пользовательской выдачи, то есть точному выбору нескольких лучших объектов, а основной мерой для оптимизации была выбрана ARP1.

Для сравнительных тестов были использованы два набора данных. Первый – DenseNetflix, часть набора Netflix, подматрица с плотностью 4.4%, содержащая 2.5 млн. оценок, 1024 объектов и 56000 пользователей, сформированная путем фильтрации объектов и пользователей с низким числом оценок. Второй – MailRuApps, набор действий, совершенных пользователями с приложениями в социальной сети. В выделенном плотном ядре содержатся около 22 млн. оценок, составленных из 400 млн. действий, выполненных 545 тыс. пользователей над 3300 приложений.

Также для сравнения скорости работы алгоритмов использовался набор данных MailRuMusic, содержащий 155 млн. неявных оценок, 2.46 млн. пользователей и 295 тыс. объектов.

В приведенной таблице показаны меры качества и время работы для коллекции DenseNetflix. RMSE значительно выше, чем при расчете по полному набору данных Netflix, это объясняется большей плотностью матрицы.

| DenseNetflix | RMSE | ARP1 | Время(с) |
|---|---|---|---|
| K=5 | 0.8782 | 0.2784 | 7.1 |
| K=10 | 0.8527 | 0.2479 | 11.2 |
| K=20 | 0.8390 | 0.2298 | 18.6 |
| K=50 | 0.8232 | 0.2156 | 35.8 |
| K=100 | 0.8211 | 0.2109 | 69.3 |

Ниже сравниваются алгоритмы сортировки и предварительного отбора объектов на массиве данных MailRuApps. Видно, что на этом наборе во всех случаях лучше работает сортировка без учета смещения оценок. Также можно заметить, что LSH4 работает с качеством, близким к полному перебору объектов, но при этом быстрее. Тем не менее, объектов в этой коллекции немного и скорость работы полного перебора еще достаточно велика.

| MailRuApps | ARP1 | Время(мс) |
|---|---|---|
| Full-Bias | 0.1671 | 34 |
| Full-SP | **0.1352** | |
| LSH1-Bias | 0.1896 | 4 |
| LSH1-SP | 0.1529 | |
| LSH4-Bias | 0.1756 | **11** |
| LSH4-SP | **0.1412** | |
| UB-Bias | 0.1783 | 28 |
| UB-SP | 0.1407 | |

На масштабе коллекции MailRuMusic уже заметна существенная разница в скорости работы между полным перебором и LSH.

| MailRuMusic | ARP1 | Время(мс) |
|---|---|---|
| Full-SP | **0.4512** | 2318 |
| LSH1-SP | 0.6874 | 9 |
| LSH4-SP | **0.4908** | **32** |
| UB-SP | 0.4683 | 1789 |

Увеличение числа плоскостей в LSH приведет к уменьшению размера сектора, а, значит, и среднего числа объектов в одном секторе. Это ускорит работу LSH, но снизит качество. Таким образом, число плоскостей подбирается как баланс между качеством и скоростью работы.

## 9. Выводы

В результате, взяв в виде базы алгоритм BRISMF, мы получили систему, которая, помимо решения задачи MF, оптимизирует настраиваемые параметры алгоритма, причем учитывает при оптимизации, помимо качества, стабильность результатов и скорость сходимости. Кроме того, был разработан механизм конвертации неявных

данных в явные оценки, что значительно расширяет спектр алгоритмов, применяемых для анализа наборов неявных данных. Система показала высокую скорость работы: на полном наборе данных Netflix с $K = 50$ уходит около 12 минут при работе в один поток.

Также было исследовано применение метода случайных проекций LSH для ускорения формирования выдачи рекомендаций. Метод показал хорошие результаты работы и может быть рекомендован к использованию, особенно при работе с большим количеством объектов.

Архитектура системы позволяет как пересчитывать вектора профилей пользователей и объектов, обеспечивая при этом постоянную работоспособность, так и формировать рекомендации для новых пользователей на лету, без полного пересчета матриц профилей.

## Литература


[1] Adomavicius G., Tuzhilin A.: Toward the next generation of recommender systems: A survey of the state-of-the-artand possible extensions. IEEE Transactions of Knowledge and Data Engineering 17(6), pp. 734-739 (2005)

[2] Bennett J., Lanning S. The Netflix Prize. In Proc. of KDD Cup Workshop at SIGKDD-07, 13th ACMInt. Conf. on Knowledge Discovery and DataMining, pages 3–6, San Jose, California, USA, 2007.

[3] Datar M., Immorlica N., Indyk P., Mirrokni V.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceeding SCG '04 Proceedings of the twentieth annual symposium on Computational geometry, 2004. ISBN:1-58113-885-7

[4] Funk S.: Netflix Update: Try This At Home. http://sifter.org/~simon/journal/20061211.html, 2006.

[5] Goldberg D., Nichols D., Oki B.M., Terry D.: Using collaborative filtering to weave an information tapestry. In: Communications of the ACM - Special issue on information filtering Volume 35 Issue 12, Dec. 1992

[6] Hannon J., Bennett M., Smyth B.: Recommending Twitter Users to Follow Using Content and Collaborative Filtering Approaches. In: RecSys '10 Proceedings of the fourth ACM conference on Recommender systems (2010)

[7] Hu Y., Koren Y., Volinsky C.: Collaborative filtering for implicit feedback datasets. In ICDM-08, 8th IEEE Int. Conf. on Data Mining, pages 263–272, Pisa, Italy, 2008.

[8] Koren Y., Ave P., Park F.: Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model. In: KDD '08 Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (2008)

[9] Pan R., Zhou Y., Cao B. et al. One-Class Collaborative Filtering. In: 2008 Eighth IEEE International Conference on Data Mining, pp. 502-511

[10] Pazzani M.J. and Billsus D. Content-based recommendation systems. In The Adaptive Web, pages 325–341. Springer Verlag, 2007.

[11] Pilászy I., Zibriczky D., Tikk D.: Fast ALS-based Matrix Factorization for Explicit and Implicit Feedback Datasets Categories and Subject Descriptors In: RecSys '10 Proceedings of the fourth ACM conference on Recommender systems (2010)

[12] Takács G., Pilászy I., Németh B., Tikk D.: Major components of the Gravity recommendation system. In: ACM SIGKDD Explorations Newsletter - Special issue on visual analytics. Volume 9 Issue 2, December 2007

[13] Takács G., Pilászy I., Németh B., Tikk D.: Scalable Collaborative Filtering Approaches for Large Recommender Systems. In: The Journal of Machine Learning Research Volume 10, 12/1/2009


## Architecture of Recommender System with Implicit User Feedback

© A.N. Fedorovsky, V.K. Logacheva

Recommender systems are the brand new type of service which aim is to distinguish user tastes based on her explicit ratings of objects or other actions with them which can be considered as implicit ratings. This paper describes constructing of such system. The main features considered besides performance include working speed, results stability and operating with sufficiently large implicit datasets. System architecture, used methods and quality measures are given in details.