УДК 004.85

Построение коллаборативной рекомендательной системы на основе сингулярного разложения матрицы

Гришанов Я.И., Пономарев И.В.
Алтайский государственный университет, г. Барнаул
me1stmott1e@gmail.com, igorpon@mail.ru

Аннотация

В данной статье описывается процесс реализации такого метода машинного обучения, как рекомендательная система; рассматривается построение коллаборативной рекомендательной системы, в основе которой лежит алгоритм сингулярного разложения или сингулярной декомпозиции матрицы. Описаны процесс сбора тестовых данных, их обработки, а также обучение модели и её оценка согласно некоторым метрикам.

Ключевые слова: сингулярное разложение, SVD, рекомендательная система, машинное обучение, python, numpy, API.

1. Введение

Одним из инновационных подходов к построению коллаборативных рекомендательных систем является использование сингулярного разложения матрицы (Singular Value Decomposition, SVD). Этот метод не только позволяет эффективно обрабатывать большие объемы данных, но и обеспечивает высокую точность и персонализацию рекомендаций. В данной статье мы рассмотрим основы построения коллаборативной рекомендательной системы на основе сингулярного разложения матрицы. Мы разберем ключевые концепции, преимущества данного подхода и его практическое применение в создании интеллектуальных систем рекомендаций.

2. Сбор данных

В качестве данных, с которыми будет работать построенная система используется открытая информация пользователей в социальной сети ВКонтакте – отметка “Мне нравится” и “Поделиться записью”, а сами пользователи, данные которых будут собираться – подписчики официального сообщества института математики и информационных технологий.

Для сбора данных используется API. API, или интерфейс программирования приложений, представляет собой набор правил и инструментов, которые позволяют различным программам взаимодействовать между собой. Это своего рода мост, который позволяет одной программе использовать функции или данные другой программы, не раскрывая ее внутреннюю реализацию. API обычно состоит из двух основных компонентов: точек доступа, они же endpoints, и набора правил взаимодействия. Точки доступа представляют URL-адреса, по которым приложения могут отправлять запросы и получать ответы. Правила взаимодействия определяют формат этих запросов и ответов. Одним из примеров API может быть веб-API, где приложение отправляет HTTP-запросы к серверу через определенные URL-адреса, а сервер возвращает данные в установленном формате, таком как JSON или XML.

3. Математические основы рекомендательных систем

В данном параграфе приведем математический аппарат, необходимый для программной реализации рекомендательной системы.

1. Сингулярное разложение

Пусть матрица A порядка $m \times n$ состоит из элементов поля K, где K – либо поле вещественных чисел, либо поле комплексных чисел. Тогда сингулярным разложением матрицы A является разложение следующего вида

$$A = U\Sigma V^T,$$

где U – квадратная ортогональная матрица размерности $m \times m$, V – квадратная ортогональная матрица размерности $n \times n$, а Σ – диагональная матрица размерности $m \times n$, состоящая из неотрицательных элементов, причем

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_k, \quad k = \min\{m, n\}.$$

В случае разреженной матрицы, можно использовать усеченное разложение, в котором остаются лишь d первых чисел σ, а остальные полагаются равными нулю. Размерность d – это гиперпараметр, отвечающий за количество скрытых свойств у пользователей и групп:

$$A'_{m\times n} = U'_{m\times d}\Sigma'_{d\times d}V'^{T}_{d\times n},$$

где в нижних индексах указаны размерности матриц.

Полученная матрица A' хорошо приближает исходную матрицу A. Более того, является наилучшим низкоранговым приближением с точки зрения среднеквадратичного отклонения.

2. Градиентный спуск

Численный метод нахождения локального минимума или максимума функции с помощью движения вдоль градиента, один из основных численных методов современной оптимизации.

Пусть целевая функция имеет следующий вид:

$$F(\overline{x}) : \; X \; \rightarrow \; R$$

и задача оптимизации задана следующим образом:

$$\min_{\overline{x}\in X} F(\overline{x}).$$

Основная идея метода заключается в том, чтобы идти в направлении наискорейшего спуска, то есть по направлению, которое задается антиградиентом $-\nabla F(\overline{x})$:

$$\overline{x}^{|j+1|} = \overline{x}^{|j|} - \lambda^{|j|}\nabla F\left(\overline{x}^{|j|}\right),$$

где $\lambda^{|j|}$ (learning rate) задает скорость градиентного спуска, этот параметр очень важен для успешного обучения модели. Если learning rate слишком большой, модель может осциллировать или не сойтись к оптимальному решению.

3. Функция потерь

Функция, которая в теории статистических решений характеризует потери при неправильном принятии решений на основе наблюдаемых данных называется функцией потерь. В статистике функция потерь обычно используется для оценки параметров моделей, а рассматриваемое событие является разностью между оцененным и истинным значениями

для каждого наблюдения набора данных. Существует несколько видов функций потерь, например, квадратичная или MSE (Mean Squared Error)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \widehat{y}_i\right)^2.$$

4. Регуляризация

Регуляризатор является дополнительным членом, добавляемым к функции потерь, чтобы контролировать сложность модели или предотвращать переобучение. Регуляризатор вводит дополнительные ограничения или штрафы за сложность модели, помогая сбалансировать точность на обучающих данных и способность модели работать на новых, ранее не виденных данных. Этот подход способствует созданию более устойчивых и эффективных моделей, повышая их способность к обобщению.

5. L_2-регуляризация

Пусть для эмпирического риска Q: Lявляется функцией потерь, β – вектором параметров $g(x,\beta)$ из модели алгоритма, а λ – неотрицательный гиперпараметр, являющийся коэффициентом регуляризации, тогда

$$Q\left(\beta, X^l\right) = \sum_{i=1}^{l} L\left(y_i, g\left(x_i, \beta\right)\right) + \lambda\sum_{j=1}^{n}\beta_j^2.$$

4. Использование математических методов в парадигме рекомендательной системы

Пусть имеется матрица оценок R, сделаем ее сингулярное разложение. Определим гиперпараметр d, он соответствует выбору первых d столбцов в матрице U и первых d строк в матрице V^T, это исходит из того, что матрица Σ' после подматрицы $k \times k$ содержит только нули. Матрицы U' и Σ' перемножим, и получим матрицу $m \times d$. В итоге получится такое усеченное разложение:

$$R \approx U_{m\times d} \times V^T_{d\times n}.$$

$$\widehat{r}_{ui} = \left(p_u, q_i\right),$$

где $\widehat{r}_{ui}$ – предсказанный рейтинг для группы, полученный при перемножении матриц U и V^T, а p_u и q_i – их соответствующие вектора. Другими словами, теперь каждый пользователь представлен строкой из матрицы U, а каждая группа представлена столбцом из матрицы V^T и рейтинг, который поставит пользователь u сообществу i приближается через скалярное произведение векторов p_u и q_i. Таким образом можно приблизить исходную матрицу оценок. Однако проблема в том, что исходная матрица R разреженная, и необходимость встает именно в том, чтобы заполнить нулевые элементы приближенными оценками. Для этого необходимо приблизить уже имеющиеся оценки числами, полученными в результате перемножения матриц U и V^T. Таким образом можно получить и оценки для нулей у исходной матрицы R, то есть встает задача минимизации следующей функции потерь:

$$L = \frac{1}{n}\sum_{u,i}\left(r_{ui} - \widehat{r}_{ui}\right)^2 = \frac{1}{n}\sum_{u,i}\left(r_{ui} - \left(p_u, q_i\right)\right)^2 \to \min,$$

где суммирование идет по индексам u, i ненулевых элементов исходной матрицы R, а n – количество оценок в матрице.

Теперь, чтобы избежать возможного переобучения, добавим слагаемое регуляризации $\lambda\left(\Sigma_u p_u^2 + \Sigma_i q_i^2\right)$, и получаем:

$$L = \frac{1}{n}\sum_{u,i}\left(r_{ui} - (p_u, q_i)\right)^2 + \lambda\left(\Sigma_u p_u^2 + \Sigma_i q_i^2\right) \to \min.$$

Осталось посчитать градиенты для реализации поиска минимума у полученной функции потерь. После расчета частных производных по p_u и q_i получаем следующие правила обновления элементов в матрицах U и V^T

$$p_{u,j} = p_{u,j} + \gamma\left(\left(r_{ui} - \widehat{r}_{ui}\right) q_{i.j} - \lambda p_{u,j}\right),$$

$$q_{i.j} = q_{i.j} + \gamma\left(\left(r_{ui} - \widehat{r}_{ui}\right) p_{u,j} - \lambda q_{i.j}\right).$$

Здесь индексы u, i – номера пользователя и группы в матрицах U (номер строки) и V^T (номер столбца), j – компонента векторов $p_{u,j}$ и $q_{i.j}$, а γ – отвечает за скорость градиентного спуска.

5. Программная реализация

Методы, описанные выше позволяют собирать необходимые для рекомендательной системы данные, однако требуется автоматизация этого процесса. В качестве языка программирования был выбран язык Python. Обосновать выбор можно его простотой, а также множеством уже реализованных библиотек для работы с удаленными серверами. В качестве среды для разработки был выбран текстовый редактор Visual Studio Code и Google Collaboratory. Данный выбор обусловлен высокой гибкостью и удобностью рабочих сред. Описываемый далее программный код находится по ссылке https://drive.google.com/drive/folders/1AF4R7lHSWq4uWCeWMe9Wh_fq2fBV7KQk?usp=sharing.

Для начала работы алгоритма необходимо подключить следующие библиотеки:

– за работу с удаленным сервером отвечает библиотека requests, она обладает необходимым методом get, который позволит отправлять GET-запросы на удаленный сервер VK с компьютера;

– библиотека json, отвечающая за методы для десериализации JSON объектов.

Единожды отправляется запрос на сервер, после чего значение ключа записывается в переменную user_token. Помимо ключа доступа в переменную version_API записывается используемая версия API, а в переменную domain короткий адрес сообщества ИМИТ.

Создаем переменную user_list, это будет список всех участников официального сообщества ИМИТ. С помощью метода .get у объекта requests отправляем GET-запрос на сервер VK. Указываем необходимые параметры. Ответом на запрос придет JSON файл, при помощи подключенной ранее библиотеки сразу же десериализуем его и вычленяем список участников, который находится в ключе items. В переменной user_list находится список, где $\{0, \ldots, N-1\}$ – номер пользователя. Далее создаём пустой словарь users, у него будет реализована следующая структура: Уникальный идентификатор пользователя будет выступать в роли ключа, а список сообществ, на которые подписан пользователь будут являться значением, доступным по ключу. После создания, пустой словарь итеративно заполняется. При этом, могут возникнуть трудности, к примеру, если у пользователя закрытый профиль, то при десериализации в JSON файле будет отсутствовать ключ items, и возникнет исключение. Чтобы предотвратить такой исход, используется конструкция try-except. Теперь, при возникновении подобной ситуации, исключение будет программно перехватываться, и программа продолжит работать, при этом удастся избежать пустой записи в словарь.

В переменной users находится словарь со следующей структурой:

$$UserID_i : [GroupID_0, \ldots, GroupID_{k-1}],$$

где $0 \leq i \leq N - 1$.

Далее необходимо построить таблицу и заполнить её коэффициентами. Данный коэффициент будет показывать то, насколько пользователь оценивает то или иное сообщество. Для этого была написана функция get_factor_table, в качестве аргумента она принимает словарь, в котором уникальный идентификатор пользователя выступает в качестве ключа, а список групп, на которые он подписан в качестве значения. Для начала создается пустой словарь table, в дальнейшем это и будет таблица с коэффициентами. Уникальный идентификатор пользователя будет выступать в качестве ключа, а сам коэффициент в качестве значения. Затем, в теле функции, алгоритм итеративно перебирает всех пользователей по очереди. Для каждого пользователя, в новом цикле, алгоритм перебирает список сообществ, на которые подписан пользователь. После этого объявляется переменная posts, она создана для хранения постов, сделанных сообществом. Чтобы получить список постов, при помощи метода .get отправляется запрос на сервер, в качестве параметра count указано значение равное 10, это сделано для того, чтобы ограничить количество постов, приходящих в ответе, а также для того, чтобы сохранить актуальность отметок, оставленных пользователем. Теперь создадим переменные like_counter / repost_counter, которые будут отвечать за количество отметок «Мне нравится» и репостов соответственно, а также за их общее количество для конкретного пользователя. После создания переменных, запускаем еще один цикл, но уже по каждому посту в сообществе. Здесь для каждого поста делается GET-запрос на сервер. Внутри запроса, при помощи метода isLiked проверяется наличие отметки «Мне нравится» и репоста. Далее, в случае если пользователь поставил отметку «Мне нравится» переменной like_counter и like_counter_all прибавляется единица, если же пользователь сделал репост записи, то единица прибавляется в переменные repost_counter и repost_counter_all. При каждом из запросов, может вызываться исключение, связанное персональными настройками пользователя: к примеру, в профиле может быть скрыт список сообществ, подобные случаи обрабатываются конструкцией try-except-else, в случае, когда исключения не были вызваны и каждый шаг был успешно завершен, в блоке else делается запись в переменную table. В результате функция вернет структуру, где в соответствие уникальному идентификатору пользователя ставится коэффициент, определяющий оценку сообщества пользователем. Переменная table это список со следующей структурой:

$$''UserID_i'' :'' Strength_{ij}''$$

где i – номер пользователя, а j – номер оценки группы пользователем.

Импортируем две библиотеки: numpy и pandas. Numpy в основном пригодится благодаря новому типу данных array и методам, которые он поддерживает, что позволит сделать работу более эффективно. Pandas будет использоваться для построения таблицы, которая наглядно отобразит имеющиеся данные.

Далее считаем полученный ранее файл users.csv и сформируем на его основе таблицу и библиотеки Pandas. Выведем полученную таблицу. В столбце user_id находится уникальный идентификатор пользователя, в столбце group_id уникальный идентификатор, а strength это численный показатель, отвечающий за количество действий пользователя в сообществе, исходя из которых будем считать, что чем больше значение в strength, тем более предпочтительно сообщество пользователю.

На основании таблицы df сформируем сводную таблицу таблицу, где по горизонтали будут находиться сообщества, по вертикали пользователи, а на пересечении некий рейтинг оценки сообщества пользователем.

	user_id	group_id	strength
0	7098730	31480508	0
1	7098730	38551279	0
2	7098730	15756023	10
3	7098730	211634919	4
4	7098730	32615394	3
...	...	...	...
588	166396401	85953128	9
589	166396401	210923985	4
590	166396401	211886381	8
591	166396401	211213079	1
592	166396401	45340576	10

Рисунок 1. DataFrame структура

Можно заметить, что многие элементы таблицы равны нулю. Это связано с тем, что оценка стоит лишь для сообществ, на которые пользователь подписан и задачей системы рекомендаций является предложить новые, наиболее релевантные сообщества, ожидая, что пользователь на них подпишется. Таким образом считается, что если пользователь не подписан на сообщество, то и его оценка равна нулю.

group_id	4846	41063	73476	77521	99052	108468	276219	558956	809187	1194208	...
user_id											
136779826	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	7.0	..
154561439	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	..
41761865	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	..

3 rows × 543 columns

Рисунок 2. Структура Pivot Table

Функция SVD реализует сингулярное разложение матрицы, при этом заполняет нулевые элементы исходной матрицы приближенными значениями с помощью алгоритма стохастического градиентного спуска. На вход функции поступает пять аргументов:

1. R – исходная матрица рейтингов;
2. d – количество свойств, которыми будет описан пользователь;
3. step – скорость градиентного спуска;
4. lambda_reg – параметр регуляризации;
5. n_iters – количество шагов, которые сделает градиентный спуск

В качестве результата, функция возвращает четыре значения: матрицы U и V^T, а также изначальную ошибку и ошибку на каждой итерации.

Для начала инициализируем матрицы U и V^T для разложения и заполняем их средним значением рейтинга в исходной матрице R. Проходя по ненулевым индексам в матрице R запишем их в список, а также посчитаем изначальную ошибку. После чего выполним уточнение значений по полученным ранее формулам обновления шага при помощи стохастического градиентного спуска, разница с обычным градиентным спуском лишь в том, что градиент оптимизируемой функции считается на каждом шаге не как сумма градиентов от каждого элемента выборки, а как градиент от одного, случайно выбранного элемента. В конце посчитаем ошибку от полученных результатов.

Вызываем функцию, передавая в нее гиперпараметры $d = 3$, $step = 0.005$, $lambda\_reg = 0.11$ и $n\_iters = 100000$. Подбор гиперпараметров является важной задачей в машинном обучении и может существенно влиять на производительность и точность модели. Один из способов подбора – это поиск по сетке. Сеточный поиск – это метод, при котором задается множество значений для каждого гипер параметра, и модель обучается и оценивается для каждой комбинации значений. Затем выбирается комбинация, которая показывает наилучшую производительность на валидационном наборе данных. Этот подход может быть вычислительно затратным, но позволяет исследовать широкий диапазон значений гипер параметров.

Перемножим матрицы U и V^T, тем самым получим приближение R_cap к исходной матрице R. Выведем полученную матрицу.

group_id	4846	41063	73476	77521	99052	108468	276219	558956	809187	1194208	...
user_id											
7098730	1.567498	5.143761	5.381350	5.385188	5.143761	4.694747	7.256723	1.250993	6.266738	4.905882	...
8208521	1.466912	6.456589	6.947031	6.884017	6.456589	5.911885	9.540140	1.189315	8.134555	6.210794	...
9698448	0.964656	4.163241	4.452998	4.403084	4.163241	3.800368	6.106125	0.751697	5.200940	3.995805	...
12604198	2.620517	6.488958	6.559262	6.661958	6.488958	5.908144	8.627000	2.094227	7.591900	6.127369	...
13677210	4.103653	7.899657	7.653227	7.913467	7.899657	7.168667	9.743921	3.273133	8.785864	7.369881	...
18058011	2.417391	6.546954	6.700820	6.771260	6.546954	5.967211	8.893139	1.934375	7.774020	6.204562	...
22855475	2.594388	6.590061	6.687052	6.782207	6.590061	6.002661	8.818924	2.075731	7.745877	6.229857	...

Рисунок 3. Приближение к матрице R

Получим список наиболее релевантных сообществ для одного из пользователей. Для этого найдем те сообщества, на которые пользователь уже подписан и удалим их из всего списка сообществ, а для тех что остались отсортируем значение по убыванию (см. рисунок 4).

```
group_id
15756023     9.886507
45340576     9.855050
147923770    9.823942
215683575    9.670190
40553536     9.380913
Name: 68686982, dtype: float64
```

Рисунок 4. Наиболее релевантные сообщества для заданного пользователя

Выведем результат для заданного пользователя по всем сообществам, на которые он подписан и сравним предсказанные рейтинги с теми, которые были в матрице R изначально и сравним результаты.

```
user: 68686982 | group: 63657636 | initial: [5] | predict: 4.903739431105686
user: 68686982 | group: 10175642 | initial: [2] | predict: 2.261746893647172
user: 68686982 | group: 194135791 | initial: [3] | predict: 3.1992888557826964
user: 68686982 | group: 186208863 | initial: [8] | predict: 7.638982455403698
user: 68686982 | group: 155790705 | initial: [8] | predict: 7.577360751363707
user: 68686982 | group: 25205856 | initial: [4] | predict: 4.044462101833352
user: 68686982 | group: 204636234 | initial: [2] | predict: 2.23865983422313
user: 68686982 | group: 154033293 | initial: [5] | predict: 4.916252205445299
user: 68686982 | group: 213945978 | initial: [4] | predict: 4.00960580773498
user: 68686982 | group: 140355142 | initial: [10] | predict: 9.390511385668614
user: 68686982 | group: 162688700 | initial: [0] | predict: 5.211851610638027
user: 68686982 | group: 74867418 | initial: [1] | predict: 1.398869908560349
user: 68686982 | group: 137331585 | initial: [6] | predict: 5.815393194464161
user: 68686982 | group: 147697500 | initial: [1] | predict: 1.4507047606910168
user: 68686982 | group: 187018433 | initial: [7] | predict: 6.708312905534916
user: 68686982 | group: 171096681 | initial: [6] | predict: 5.807281547870292
user: 68686982 | group: 206596479 | initial: [4] | predict: 4.008773967654071
user: 68686982 | group: 52846065 | initial: [3] | predict: 3.1434623622183127
user: 68686982 | group: 4846 | initial: [1] | predict: 1.3934082870062081
user: 68686982 | group: 213921767 | initial: [6] | predict: 5.82590648023134
user: 68686982 | group: 191718805 | initial: [6] | predict: 5.823479005229015
user: 68686982 | group: 199104221 | initial: [6] | predict: 5.829487887292327
user: 68686982 | group: 188019044 | initial: [4] | predict: 4.023816036557004
user: 68686982 | group: 65985923 | initial: [2] | predict: 2.2431734239581127
user: 68686982 | group: 88200327 | initial: [1] | predict: 1.2556803445114824
user: 68686982 | group: 102664517 | initial: [6] | predict: 5.829891029095317
user: 68686982 | group: 198484022 | initial: [10] | predict: 9.349878956286314
```

Рисунок 5. Сравнение рейтингов

Оценка initial означает изначальный рейтинг, а predict то, что предсказал алгоритм рекомендаций.

Теперь воспользуемся U-критерием Манна-Уитни чтобы оценить различия между выборками. Сформулируем нулевую гипотезу H_0: выборки равны и альтернативную H_a: выборки не равны. Выберем стандартный уровень значимости $p = 0,05$. Готовая функция уже реализована в библиотеке SciPy, воспользуемся ею. Функция возвращает следующую информацию:

```
MannwhitneyuResult(statistic=6939.0, pvalue=0.6396186310426437)
```

Рисунок 6. Результат U-test

Заметим, что pValue, которое вернула функция больше, чем заданный уровень значимости p, следовательно мы принимаем нулевую гипотезу и делаем вывод о том, что выборки равны.

6. Заключение

В результате проведенных исследований мы видим, что этот подход предоставляет мощный инструмент для предсказания пользовательских предпочтений в контексте построения персонализированных рекомендаций. Методика сингулярного разложения матрицы демонстрирует свою эффективность не только в точности предсказаний, но и в способности обрабатывать большие объемы данных, что делает ее важным инструментом в области разработки интеллектуальных систем.

Этот подход не только улучшает уровень удовлетворения пользователей, предоставляя им контент, соответствующий их интересам, но также открывает новые возможности для более глубокого понимания структуры данных и взаимодействий в онлайн-средах.

Вмешательство сингулярного разложения матрицы в область рекомендательных систем предоставляет инженерам и исследователям новый инструментарий для создания более умных и адаптивных технологий.

Однако, несмотря на все достижения, важно продолжать исследования в этой области, стремиться к новым методам оптимизации и совершенствованию алгоритмов. Только таким образом мы сможем эффективно справляться с вызовами быстро меняющегося цифрового ландшафта и создавать рекомендательные системы, которые не только предсказывают, но и понимают потребности и предпочтения каждого пользователя.

Список литературы

1. Рекомендательные системы на практике / пер. с англ. Д.М. Павлова. — М. : ДМК Пресс, 2020. — 448 с.
2. Ricci F. Recommender Systems Handbook. — New York : Springer New York Dordrecht Heidelberg London, 1979. — 845 p.
3. Habr : сайт. — URL: https://habr.com/ru/company/jetinfosystems/blog/453792/.
4. Habr : сайт. — URL: https://habr.com/ru/company/prequel/blog/567648/.
5. VK: сайт. — URL: https://dev.vk.com/reference.
6. Visual Studio Code: сайт. — URL: https://code.visualstudio.com/.
7. Retailer: сайт. — URL: https://retailrocket.ru/blog/shest-podhodov-k-rekomendatsiyam-tovarov-v-internet-magazine/.
8. Google Colab: сайт. — URL: https://colab.research.google.com/.
9. Jim Lambers. The SVD Algorithm. — 2010. — CME 335.
10. Логинов Н.В. Сингулярное разложение матриц. — М. : МГАПИ, 1996.
11. Гасников А.В. Современные численные методы оптимизации. Метод универсального градиентного спуска : учебное пособие. — М. : МФТИ, 2018. — 291 с.
12. Тихонов А.Н., Арсенин В.Я. Методы решения некорректных задач. — М. : Наука, 1979. — 283 с.
13. SciPy Stats: сайт. — URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mannwhitneyu.html.