

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА**

**по дисциплине**

**«Разработка обеспечивающих подсистем систем поддержки принятия решений»**

Студент группы: ИКБО-04-19 Писаревский А.А. *(Фамилия студента)*

Руководитель работы: Железняк Л.М.

*(Фамилия преподавателя)*

|  |  |  |
| --- | --- | --- |
|  |  |  |

Москва 2022

1. **Сингулярное разложение (SVD)**

Начнем с самого простого и понятного метода — SVD. Мы можем записать все наши данные в матрицу 𝐴 размером 𝑛×𝑚 , где 𝑛=|𝑈| , а 𝑚=|𝐼|. Для известных пар из D мы запишем элемент матрицы , а все остальные приравняем к нулю. Используя SVD разложение, можно представить матрицу 𝐴 в виде произведения трех других: 𝑈, Σ, 𝑉 . Оставив 𝑘 самых больших по модулю сингулярных значений, можно сделать малораноговую аппроксимацию нашей матрицы 𝐴.



Обозначим матрицу 𝑄 и 𝑃 следующим образом. Тогда матрица 𝐴 представляется в виде такого произведения:



Или поэлементно это можно записать:



Получается, что  и  — это латентные представления пользователя 𝑢 и товара 𝑖 в каких-то пространствах размерности 𝑘. На самом деле мы можем представить нашу модель в конечном виде и сразу искать латентные представления товаров и пользователей. Для этого просто поговорим немного об оптимизации. В данной задаче параметрами выступают латентные вектора:



И в этом случае они будут находиться при минимизации квадратичной ошибки c учетом регуляризации:

****

Сразу поговорим про улучшение этого метода. В реальности бывает так, что некоторые люди склонны завышать оценки товаров, а другие наоборот занижать. Так же и сами товары могут быть довольно популярны и менее популярны. Поэтому обычная модель SVD работает не так хорошо. Для решения этой проблемы нужно просто использовать обычное смещение (bias):



где  — смещение для конкретного пользователя,  — для товара, а — глобальное смещение. При этом количество параметров немного вырастет:



1. **SVD++**

В работе Factorization Meets the Neighborhoo[[1]](#footnote-1)описывается новая модификация SVD модели. Но для начала поговорим про явный и неявный отклик от пользователя (explicit and implicit user feedback). Если по итогу взаимодействия между пользователем и товаром мы знаем оценку 𝑟𝑢𝑖 , то это считается явным откликом. В противном случае мы знаем лишь об их взаимодействии и это неявный. Авторы статьи ассоциируют каждого пользователя с двумя группами товаров: 𝑅(𝑢) — множество товаров с явным откликом (рейтинги которых мы знаем) и 𝑁(𝑢) — множество товаров с неявным откликом (знаем лишь о наличии взаимодействия).

Метод SVD++ использует неявный отклик и в результате модель выглядит следующим образом:



В этом случае количество параметров увеличивается:



Поскольку неявный отклик иногда бывает недоступным, множество 𝑁(𝑢) можно заменить на 𝑅(𝑢) , т.к. всегда выполняется 𝑅(𝑢)⊂𝑁(𝑢).

Добавку данного метода можно расценивать как рекомендации товара к товару (item-item recommendation).

1. **Реализация и результат применения SVD++**

Реализация алгоритма SVD++ была осуществлена с помощью языка программирования Python в среде JupyterLab.

Пусть исходная рейтинговая матрица выглядит так:

Табл. 3.1 – История приобретений пользователями товаров (1 – приобретено, 0 – не приобретено)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | О1 | О2 | О3 | О4 | О5 |
| П1 | 1 | 1 | 1 | 1 | 1 |
| П2 | 0 | 0 | 0 | 1 | 0 |
| П3 | 1 | 1 | 0 | 0 | 1 |

Тогда после применения алгоритма и вычисления показателей скрытых факторов относительно пользователей (П) и товаров (О), выводом программы будет приближенная матрица:

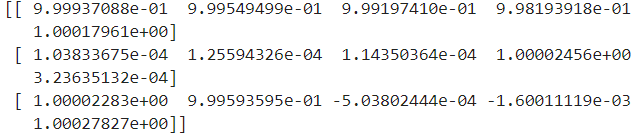


Рис. 3.1 – Результат работы программы

**Вывод**

В данной практической работе была приведена теоретическая справка об алгоритме факторизации матриц SVD и его модернизации – SVD++. В частности, был рассмотрен математический аппарат, позволяющий производить операцию факторизации матриц, что имеет практический смысл при построении рекомендательных систем, где учитываются взаимодействия пользователей с товарами.

А затем была осуществлена реализация алгоритма SVD++ для проведения факторизации пользовательской матрицы.

**Приложение**

Реализация алгоритма на языке Python 3.9

*Листинг А - Код реализации алгоритма SVD++*

|  |
| --- |
| def svdpp():  for epoch in range(epochs):  for user in range(num\_users):  for item in range(num\_items):  # activation  p\_i = p[user, :]  q\_j = q[:,item]  y\_j = y[:, item]    user\_item\_list = [i for i, e in enumerate(r[user,:]) if e != 0] # R\_u (item indices which a user interacted with)  sum\_y\_j = np.sum(y[:,user\_item\_list],axis=1) # to make sum of y\_i  r\_hat[user,item] = bu[user, item] + bi[user, item] + mu[user, item] + \  np.dot(np.transpose(q\_j), p[user, :] + pow(len(user\_item\_list), -0.5)\*sum\_y\_j)  # descent  e = r[user, item] - r\_hat[user, item]  p[user] += 2\*alpha\*(e\*q\_j - lambda\_1\*p\_i)  q[:,item] += 2\*alpha\*(e\*(p\_i + pow(len(user\_item\_list), -0.5)\*sum\_y\_j) - lambda\_1\*q\_j)  y[:, item]+= 2\*alpha\*(e\*(pow(len(user\_item\_list), -0.5)\*q\_j - lambda\_1\*y\_j))  bu[user, item] += 2\*beta\*(e - lambda\_2\*bu[user, item])  bi[user, item] += 2\*beta\*(e - lambda\_2\*bi[user, item])  mu[user, item] += 2\*beta\*(e - lambda\_2\*mu[user, item])    return r\_hat |

1. https://people.engr.tamu.edu/huangrh/Spring16/papers\_course/matrix\_factorization.pdf [↑](#footnote-ref-1)