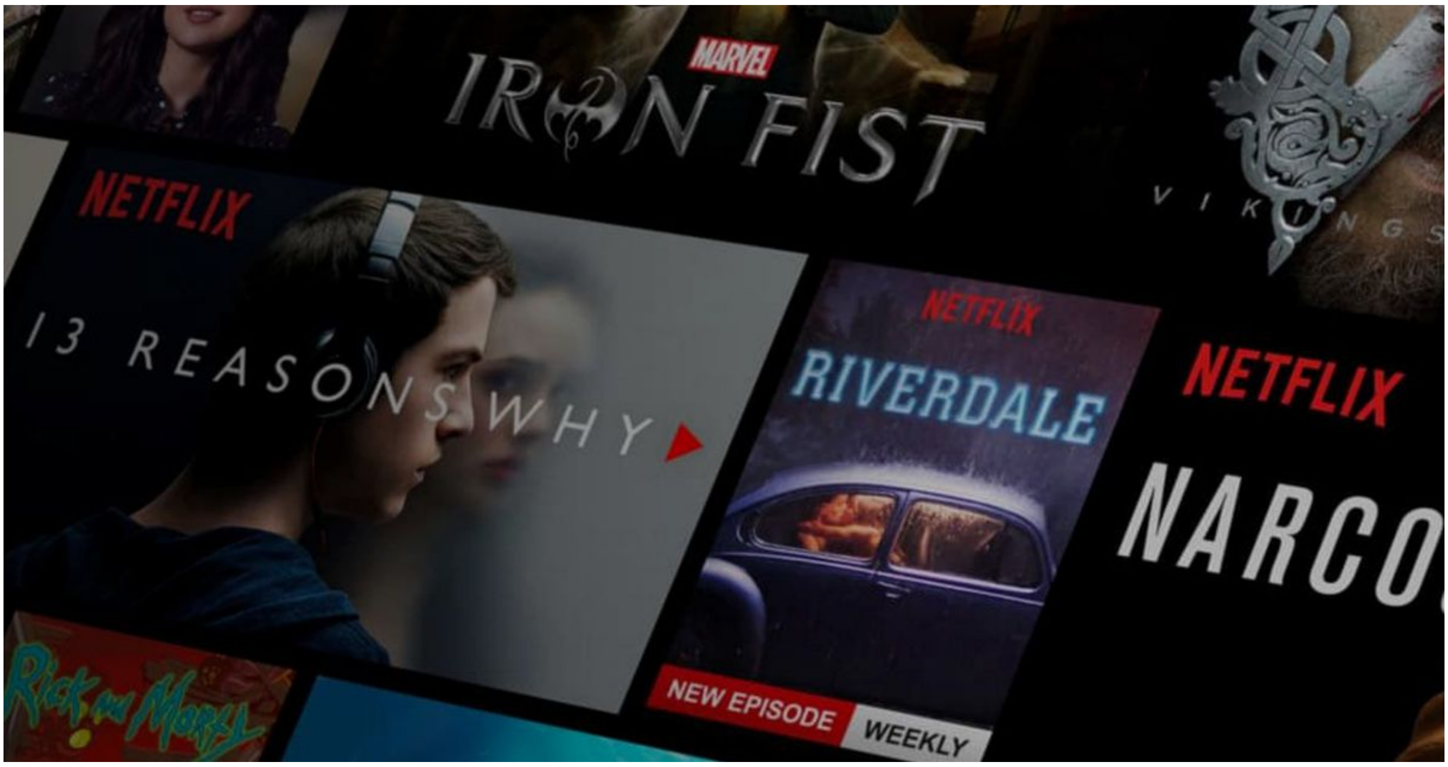
Introduction to Recommender System. Part 1 (Collaborative Filtering, Singular Value Decomposition)

Kung-Hsiang, Huang (Steeve)
Jan 28, 2018 · 8 min read



https://goo.gl/ihju1k

1. Introduction

A recommender system refers to a system that is capable of predicting the future preference of a set of items for a user, and recommend the top items. One key reason why we need a recommender system in modern society is that people have too much options to use from due to the prevalence of Internet. In the past, people used to shop in a physical store, in which the items available are limited. For instance, the number of movies that can be placed in a Blockbuster store depends on the size of that store. By contrast, nowadays, the Internet allows people to access abundant resources online. Netflix, for example, has an enormous collection of movies. Although the amount of available information increased, a new problem arose as people had a hard time selecting the items they actually want to see. This is where the recommender system comes in. This article will give you a brief introduction to two typical ways for building a recommender system, Collaborative Filtering and Singular Value Decomposition.

2. Traditional Approach

Traditionally, there are two methods to construct a recommender system :

- **Content-based recommendation**
- **Collaborative Filtering**

The first one analyzes the nature of each item. For instance, recommending poets to a user by performing Natural Language Processing on the content of each poet. Collaborative Filtering, on the other hand, does not require any information about the items or the users themselves. It recommends items based on users' past behavior. I will elaborate more on Collaborative Filtering in the following paragraphs.

3. Collaborative Filtering

As mentioned above, Collaborative Filtering (CF) is a mean of recommendation based on users' past behavior. There are two categories of CF:

- **User-based**: measure the similarity between target users and other users
- **Item-based**: measure the similarity between the items that target users rates/ interacts with and other items

The key idea behind CF is that similar users share the same interest and that similar items are liked by a user.

Assume there are *m* users and *n* items, we use a matrix with size *m*n* to denote the past behavior of users. Each cell in the matrix represents the associated opinion that a user holds. For instance, M_{i, j} denotes how user i likes item j. Such matrix is called **utility matrix**. CF is like filling the blank (cell) in the utility matrix that a user has not seen/rated before based on the similarity between users or items. There are two types of opinions, **explicit opinion** and **implicit opinion**. The former one directly shows how a user rates that item (think of it as rating an app or a movie), while the latter one only serves as a proxy which provides us heuristics about how an user likes an item (e.g. number of likes, clicks, visits). Explicit opinion is more straight-forward than the implicit one as we do not need to guess what does that number implies. For instance, there can be a song that user likes very much, but he listens to it only once because he was busy while he was listening to it. Without explicit opinion, we cannot be sure whether the user dislikes that item or not. However, most of the feedback that we collect from users are implicit. Thus, handling implicit feedback properly is very important, but that is out of the scope of this blog post. I'll move on and discuss how CF works.

User-based Collaborative Filtering

We know that we need to compute the similarity between users in user-based CF. But how do we measure the similarity? There are two options, Pearson Correlation or cosine similarity. Let u_{i, k} denotes the similarity between user i and user k and v_{i, j} denotes the rating that user i gives to item j with v_{i, j} = ? if the user has not rated that item. These two methods can be expressed as the followings:

$$u_{ik} = \frac{\sum_j (v_{ij} - v_i)(v_{kj} - v_k)}{\sqrt{\sum_j (v_{ij} - v_i)^2 \sum_j (v_{kj} - v_k)^2}}$$

Pearson Correlation (https://goo.gl/y93CsC)

$$\cos(u_i, u_j) = \frac{\sum_{k=1}^{m} v_{ik} v_{jk}}{\sqrt{\sum_{k=1}^{m} v_{ik}^2 \sum_{k=1}^{m} v_{jk}^2}}$$

Cosine Similarity (https://goo.gl/y93CsC)

Both measures are commonly used. The difference is that Pearson Correlation is invariant to adding a constant to all elements.

Now, we can predict the users' opinion on the unrated items with the below equation:

$$v_{ij}^{*} = K \sum_{v_{kj} \neq ?} u_{jk} v_{kj}$$

Unrated Item Prediction (https://goo.gl/y93CsC)

Let me illustrate it with a concrete example. In the following matrixes, each row represents a user, while the columns correspond to different movies except the last one which records the similarity between that user and the target user. Each cell represents the rating that the user gives to that movie. Assume user E is the target.

	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You	Similarity(i, E)
A	2		2	4	5		NA
B	5		4			1	
C			5		2		
D		1		5		4	
E			4			2	1
F	4	5		1			NA

Since user A and F do not share any movie ratings in common with user E, their similarities with user E are not defined in Pearson Correlation. Therefore, we only need to consider user B, C, and D. Based on Pearson Correlation, we can compute the following similarity.

	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You	Similarity(i, E)
A	2		2	4	5		NA
B	5		4			1	0.87
C			5		2		1
D		1		5		4	-1
E			4			2	1
F	4	5		1			NA

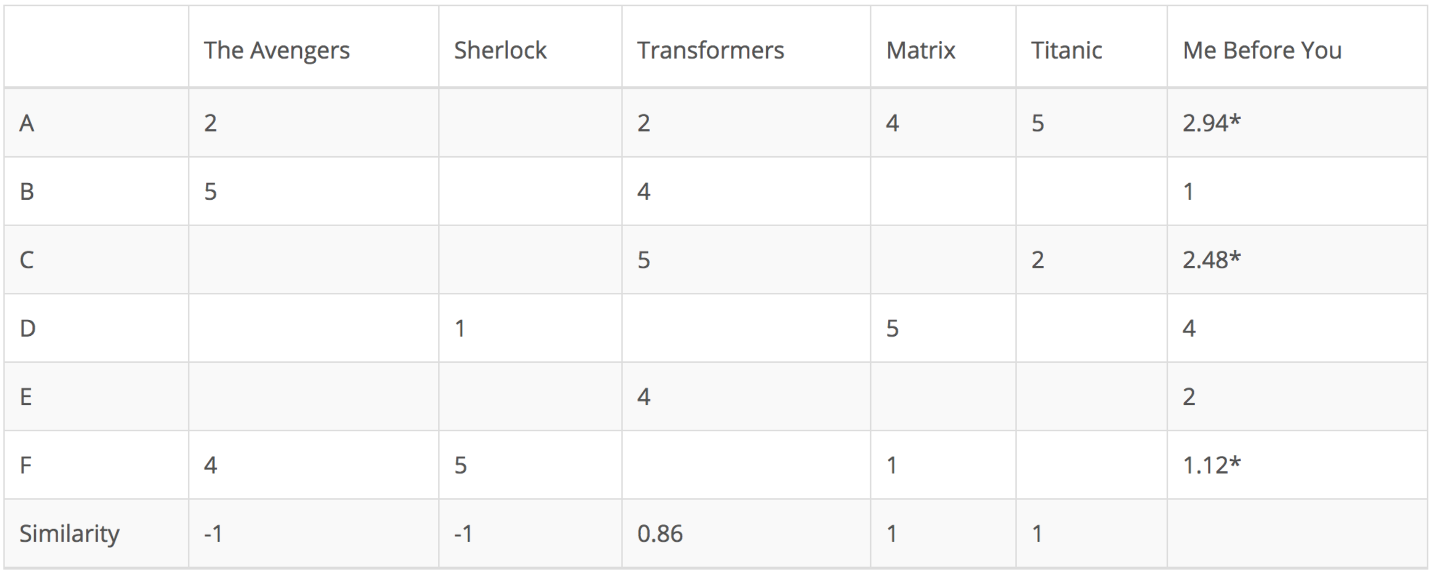
From the above table you can see that user D is very different from user E as the Pearson Correlation between them is negative. He rated *Me Before You* higher than his rating average, while user E did the opposite. Now, we can start to fill in the blank for the movies that user E has not rated based on other users.

	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You	Similarity(i, E)
A	2		2	4	5		NA
B	5		4			1	0.87
C			5		2		1
D		1		5		4	-1
E	3.51*	3.81*	4	2.42*	2.48*	2	1
F	4	5		1			NA

Although computing user-based CF is very simple, it suffers from several problems. One main issue is that users' preference can change over time. It indicates that precomputing the matrix based on their neighboring users may lead to bad performance. To tackle this problem, we can apply item-based CF.

Item-based Collaborative Filtering

Instead of measuring the similarity between users, the item-based CF recommends items based on their similarity with the items that the target user rated. Likewise, the similarity can be computed with Pearson Correlation or Cosine Similarity. The major difference is that, with item-based collaborative filtering, we fill in the blank vertically, as oppose to the horizontal manner that user-based CF does. The following table shows how to do so for the movie *Me Before You*.



	The Avengers	Sherlock	Transformers	Matrix	Titanic	Me Before You
A	2		2	4	5	2.94*
B	5		4			1
C			5		2	2.48*
D		1		5		4
E			4			2
F	4	5		1		1.12*
Similarity	-1	-1	0.86	1	1	

It successfully avoids the problem posed by dynamic user preference as item-based CF is more static. However, several problems remain for this method. First, the main issue is scalability. The computation grows with both the customer and the product. The worst case complexity is O(mn) with m users and n items. In addition, sparsity is another concern. Take a look at the above table again. Although there is only one user that rated both *Matrix* and *Titanic* rated, the similarity between them is 1. In extreme cases, we can have millions of users and the similarity between two fairly different movies could be very high simply because they have similar rank for the only user who ranked them both.

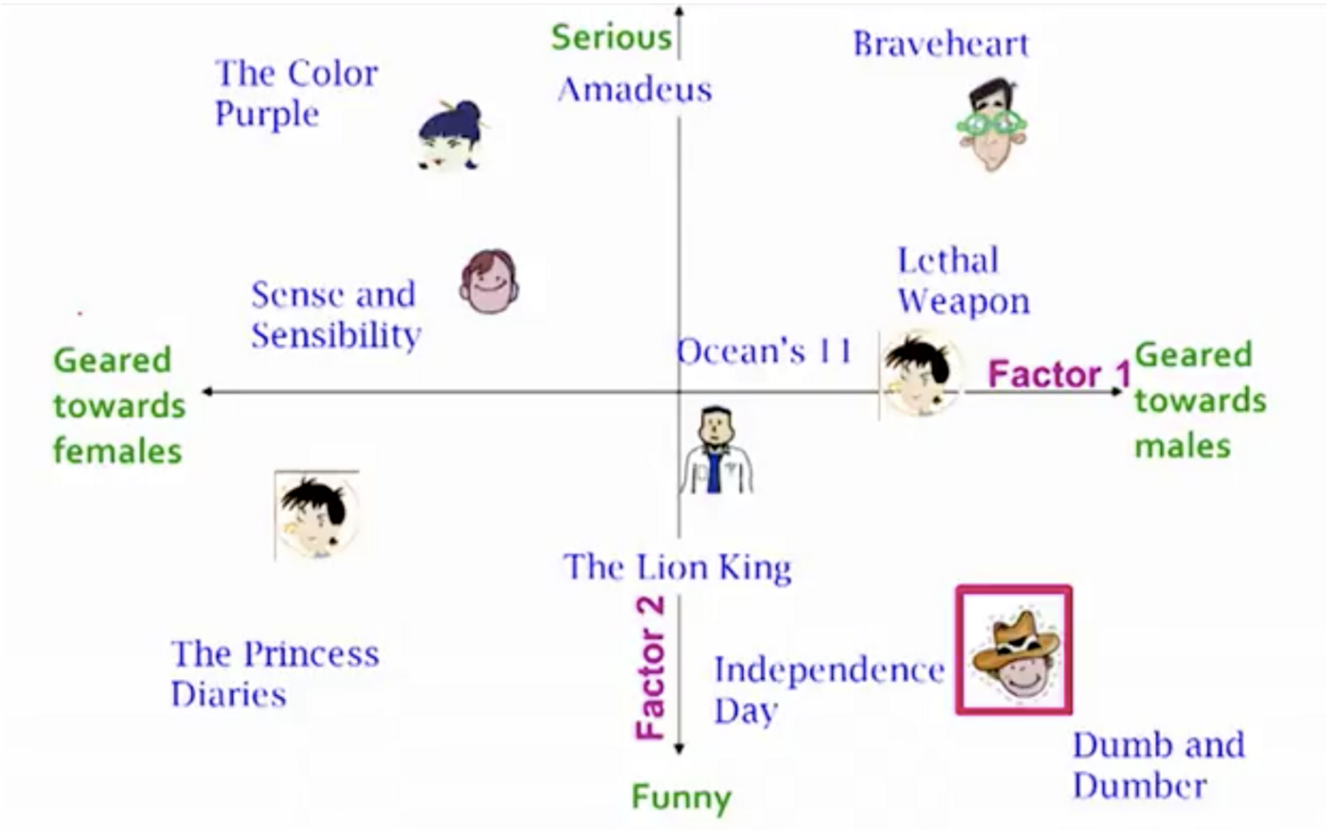
4. Singular Value Decomposition

One way to handle the scalability and sparsity issue created by CF is to leverage a **latent factor model** to capture the similarity between users and items. Essentially, we want to turn the recommendation problem into an optimization problem. We can view it as how good we are in predicting the rating for items given a user. One common metric is **Root Mean Square Error** (RMSE). The lower the RMSE, the better the performance. Since we do not know the rating for the unseen items, we will temporarily ignore them. Namely, we are only minimizing RMSE on the known entries in the utility matrix. To achieve minimal RMSE, **Singular Value Decomposition** (SVD) is adopted as shown in the below formula.

$$\overset{\hat{X}}{\underset{m\times n}{\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n}\\ x_{21} & x_{22} & \cdots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn}\end{pmatrix}}} \approx \overset{U}{\underset{m\times r}{\begin{pmatrix} u_{11} & \cdots & u_{1r}\\ \vdots & \ddots & \\ u_{m1} & & u_{mr}\end{pmatrix}}} \overset{S}{\underset{r\times r}{\begin{pmatrix} s_{11} & 0 & \cdots\\ 0 & \ddots & \\ \vdots & & s_{rr}\end{pmatrix}}} \overset{V^{\top}}{\underset{r\times n}{\begin{pmatrix} v_{11} & \cdots & v_{1n}\\ \vdots & \ddots & \\ v_{r1} & & v_{rn}\end{pmatrix}}}$$

Singular Matrix Decomposition(http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/images/svd2.png)

X denotes the utility matrix, and U is a left singular matrix, representing the relationship between users and **latent factors.** S is a diagonal matrix describing the strength of each latent factor, while V transpose is a right singular matrix, indicating the similarity between items and latent factors. Now, you might wonder what do I mean by latent factor here? It is a broad idea which describes a property or concept that a user or an item have. For instance, for music, latent factor can refer to the genre that the music belongs to. SVD decreases the dimension of the utility matrix by extracting its latent factors. Essentially, we map each user and each item into a latent space with dimension *r*. Therefore, it helps us better understand the relationship between users and items as they become directly comparable. The below figure illustrates this idea.



SVD Maps Users and Items Into Latent Space (https://www.youtube.com/watch?v=E8aMcwmqsTg&list=PLLssT5z_DsK9JDLcT8T62VtzwyW9LNepV&index=55)

SVD has a great property that it has the minimal reconstruction Sum of Square Error (SSE); therefore, it is also commonly used in dimensionality reduction. The below formula replace X with A, and S with Σ.

$$\min \sum \left(A_{ij} - [U\Sigma V^{\mathrm{T}}]_{ij}\right)^2$$

$$_{U,V,\Sigma} \sum_{ij \in A}$$

Sum of Square Error (https://www.youtube.com/watch?v=E8aMcwmqsTg&list=PLLssT5z_DsK9JDLcT8T62VtzwyW9LNepV&index=55)

But how does this has to do with RMSE that I mentioned at the beginning of this section? It turns out that RMSE and SSE are monotonically related. This means that the lower the SSE, the lower the RMSE. With the convenient property of SVD that it minimizes SSE, we know that it also minimizes RMSE. Thus, SVD is a great tool for this optimization problem. To predict the unseen item for a user, we simply multiply U, Σ, and T.

Python Scipy has a nice implementation of SVD for sparse matrix.

```
>>> from scipy.sparse import csc_matrix
>>> from scipy.sparse.linalg import svds
>>> A = csc_matrix([[1, 0, 0], [5, 0, 2], [0, -1, 0],
[0, 0, 3]], dtype=float)
>>> u, s, vt = svds(A, k=2) # k is the number of
factors
>>> s
array([ 2.75193379,  5.6059665 ])
```

SVD handles the problem of scalability and sparsity posed by CF successfully. However, SVD is not without flaw. The main drawback of SVD is that there is no to little explanation to the reason that we recommend an item to an user. This can be a huge problem if users are eager to know why a specific item is recommended to them. I will talk more on that in the next blog post.

5. Conclusion

I have discussed two typical methods for building a recommender system, Collaborative Filtering and Singular Value Decomposition. In the next blog post, I will continue to talk about some more advanced algorithms for building a recommender system. Should you have any problem or question regarding to this article, please do not hesitate to leave a comment below or drop me an email: khuangaf@connect.ust.hk. If you like this blog post, make sure you follow me on twitter for more great Deep Learning article!

Data Science Recommender Systems Collaborative Filtering

Artificial Intelligence Machine Learning